UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)

MASTER IN INNOVATION AND RESEARCH IN INFORMATICS

DATA SCIENCE

MASTER THESIS

A Benchmark for Graph Neural Networks for Computer Network Modeling

Author:
Sergi Carol Bosch

Supervisor:
Dr. Albert Cabellos - Computer Architecture
Co Supervisor:
Dra. Marta Arias - Computer Science

June 27, 2019

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

Abstract

Today, network operators still lack functional network models able to make accurate predictions of end-to-end Key Performance Indicators (e.g., delay or jitter) at limited cost. RouteNet is a Graph Neural Network that aims to provide this lacking functionality which models network schemes and is able to predict the key performance indicators. Thanks to its GNN architecture that operates over graph-structured data, RouteNet reveals an unprecedented ability to learn and model the complex relationships among topology, routing and input traffic in networks.

In previous related works it was proved that RouteNet was able to generalize multiple computer networks by training the model with various topologies of different sizes. This thesis aims to prove how RouteNet is able to model a computer network, but also, and more importantly, wants to provide the ground work for routing sampling creation, which allows to create enough routing combinations to provide a representative sample from which RouteNet would be able to be trained by only simulating a single topology. Finally myself and the research team of RouteNet would like to present the benchmark for computer network modeling using graph neural networks in the form of RouteNet (the model) and the datasets used.

Proving the two first objective would permit future research to be done much faster, since simulation and training of multiple topologies is a time costly procedure. It would also allow us to model any other network by only using the routing samples from one simulated network.

Acknowledgments

To the research team in UPC developing RouteNet for all the help they provided in developing this project, and to my two supervisors, Albert and Marta, for the help and support offered. Finally to my family and my classmates Laura, Sofia, Carles, Balbina, Carolina, Elena, and Alex for all the support.

Contents

List of Figures

List of Tables

Nomenclature

d	Delay of a link or a path.
h_l	State of a link in a computer network.
h_p	State of a path in a computer network.
l_k	A link in a computer network.
N	Computer network.
p_k	A path in a computer network.
R	Routing scheme.
x_l	Properties of a link in a computer network.
x_p	Properties of a path in a computer network.

1 Introduction

1.1 Motivation

Graph Neural Networks have been a hot topic in the recent months, with the first publication in my knowledge of the Graph Neural Network released on 9 December 2008. Yet a more recent paper by the Deep Mind team at Google [1], the use of graph neural networks to solve problems that had a deep relation with structured data, mainly graphs, has escalated greatly. Before the publication of the mentioned before paper, the modeling of systems which rallied in structured forms of data was problematic, eg: routing network systems, optimal traffic routing for cars, or chemical reactions prediction.

With the introduction of systems such as Software Define Networks (SDN) [2], and more importantly Knowledge Define Networking **KDN** [3], the paradigm of routing optimization has been made easier thanks to the simplification provided by these systems to observe and control the network, which allows traffic engineers to make better and faster decisions regarding the state of the network by using machine learning concepts to model a computer network which would allow to predict the Key Performance Metrics (*KPI*) (ej: delay and jitter), of the network. Knowledge of these metrics before setting a routing policy would have a profound impact on traffic engineering.

Therefor research on using machine learning concepts to model networking routing has been and interesting topic for many researches and companies. With this interest in mind, development towards the creation of a model that can be used to optimize or automate traffic engineering has made significant progress. These kinds of optimization's are really interesting for big companies in the market since it would mean that the burden of network and traffic engineering is to no longer relay to solely on persons but the algorithms can be the ones to decide the different combinations of options or at least can help the engineers to achieve better solutions.

The aim of this thesis will be outlined in the objectives section, but as a quick explanation the thesis objectives are to introduce a Graph Neural Network model benchmark to the problem of network routing optimization. Routing optimization as been one of the important topics for computer network engineers. A good routing policy can increase significantly the performance of any network system, and has been a deeply researched by different institutions [4] [5]. Since it is not an easy topic and there is no perfect solution for this kinds of problems due to the changing nature of internet traffic networks, specialized personnel is needed in order to create optimized routing in the networks and configure the different devices that live in the internet network appropriately.

The objective is also to prove how the creation of alternative routing combinations helps generalize any kind of topology, meaning that we do not need to

train a model with multiple topologies in order to achieve a good model performance. It also aims to provide a solution for creating a representative sample of routing schemes for a given topology which can be used to train and test the model.

1.2 Related Work

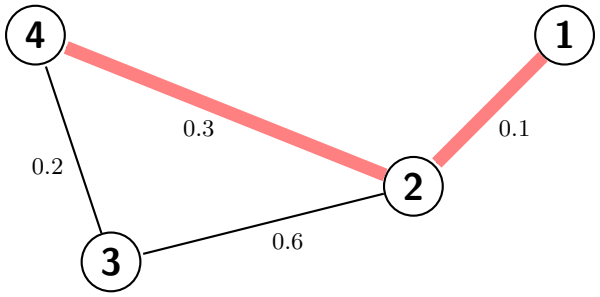
In a previous paper presented in the SIGNCOMM conference [6] myself and the research team I work with introduced a paper with the aim to challenge the generalization capabilities of Graph Neural Networks. In the mentioned paper out research team demonstrated that a Graph Neural Network is able to predict the delay of any kind of network provided that the training data comes from big enough representative sample with the network size. Yet this research was done on the basis that in order to be able to generalize a computer network, multiple topologies from different sizes were needed as training data, which then would allow to generalize any network of similar size to the training topologies size.

SIGCOMM, hosted in Beijing, China, **is one of the most, if not the most**, important conferences related to computer networks that exists, which is known to have an extremely low rate of acceptance (10%), the said paper **has been submitted and accepted** to be presented at said conference.

1.3 Key Concepts

During this thesis I will refer to a few concepts that I believe need a bit more explanation for those who are not familiar to the computer networks field. Mainly I will refer to the following concepts: topology, routing, and traffic matrix.

A topology is the graph representation of a computer network, meaning that each node is indeed a node in a network (ej: a router, server, home computer...) while the links connecting the different nodes are the physical cables connecting the computers/routers. A routing is the path taken by internet packets from one node to another node, while the traffic matrix is the bandwidth of the edge between two nodes.



The graph above shows an small example of a network topology, in which the Graph itself is the topology, the values in each weight represent the traffic

matrix and the marked red edges represent a routing between two nodes in the topology.

1.4 Objectives

The aim of this thesis is to prove the usefulness of Graph Neural Networks to accurately predict the delay of a computer network, more over instead of using a technique similar to the one done in the previous work, the paper presented in the *SIGCOMM* conference, which involved training the model with multiple topologies with different sizes, this thesis hypothesizes that this is not necessary, but instead only a representative sample of routing is needed to provide training data for a model to be able to train a Graph Neural Network that is capable to correctly generalize any kind of computer network regardless of size. To sum up, the objectives are the following:

- Create an algorithm that is able to produce a representative sample of routings from a given topology.
- Prove that in order to obtain a good model it is not needed to use different topologies sizes as train data, but only a representative sample of routings from one topology.
- Providing a benchmark for Graph Neural Networks for Computer Network Modeling.

If the thesis objectives proves satisfactory, it would mean that we are able to produce a model which is able to generalize any kind of network without the need to simulate a multitude of different sized networks, which is time consuming just to simulate, and even more time consuming in order to train a new model. This would mean that with one topology thoroughly explored we could be able to model any network no matter the size. Yet I would like to remark that the bigger the node size of the network topology, the better, since it allows for a greater exploration of the routings, as well as better generalization, since I hypothesize that our model is able to generalize any network with a lower or equal amount of nodes as the node size of the training topology.

Not only that, but it would also mean that we are able to generate a huge amount of different routings that are able to explore the different possible combinations that any network could have.

The results of this thesis will also be provided in the form of a benchmark similar to the *imagenet* benchmark [7].

1.5 Document Structure

This document will be structured in the following way: it will begin by explaining the state of the art model, which will show the theoretical point of view for **RouteNet**, then it will follow the methodology, which will explain the dataset (generation, features, and statistics) as well as the **Routing Creation algorithm** which is essential for the thesis, after the technical explanation of RouteNet architecture will be described.
With the technical and theoretical model explained, the fifth section will look at the results of the model **evaluation** and will explain them in detail.
Finally I will end the document with the conclusions of the thesis as well as some future work that this thesis opens up.

2 State of the Art

The proposed model is an *state of the art* Graph Neural Network, called **RouteNet** [8], which is able to understand the complex characteristics of a network model by interpreting it as a graph. The model is able to understand the inner structure of a network, in which the total delay of an IP packet depends on the state of the links in the network. As such, the computation of the delay has to take into account all the other packets in the links that the routing uses by calculating the state of the links and the paths of the graph.

The figure below shows the schematic representation of how the RouteNet system works, by taking as parameters the topology of the network, the routings, and the traffic matrix. The output will be the desired source-destination KPI (metrics), which is usually the delay.

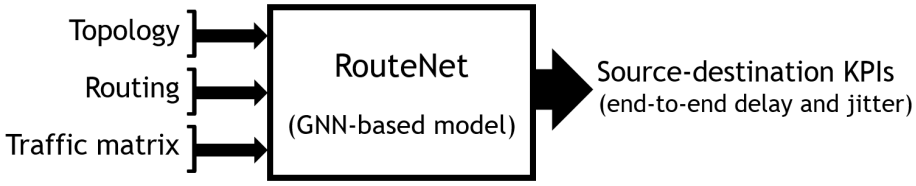


Figure 1: Schematic representation of RouteNet

2.1 Notation

A computer network is represented by a set of links $N = \{l_i\}_j$, and a routing scheme is composed by a number of different paths $R = p_k$. Each path p_k is formed by a set of links $p_k = (l_{k_1}, l_{k_2}, ..., l_{k_{(|p_k|)}})$.The properties of the paths and links are denoted $\mathbf{x}_{p_i}$ and $\mathbf{x}_{l_i}$. The state of a path is called h_{p_i}, and the state of a link is stated as h_{l_i}. d is understood as the delay of a link or a path. During the course of the explanations I will use both the terms graph and computer network indistinctly, since they are representatives of the same thing in the terms of the architecture of the Graph Neural Network.

2.1.1 Recurrent Neural Networks

I will do an small introduction to the concept of Recurrent Neural Networks due to their importance in the creation and understanding of Graph Neural Networks.
Recurrent Neural Networks have been extensively used for multiple task which require contextual information in order to understand the current input of the Neural Network, some of the examples of RNN usage are: text processing, handwriting recognition or speech recognition [9] [10]. Graph Neural Networks use RNN due to the fact that they are able to retain information from previous

states in order to update the current state of the hidden neuron, this mechanism allows to carry a bias for locality in the sequence via their Markovian structure.

2.2 Graph Neural Networks

Graph Neural Networks (**GNN**) are a recently introduced concept in the Neural Network paradigm. GNN work by comprehending the architecture of a graph, and the deep relationship that exists between nodes, as such GNN are able to understand the explicit representations of nodes and edges and also find rules for computing such relations. Graph Neural Networks have been previously successfully used to perform computations for Quantum Chemistry in order to predict quantum properties for organic molecules, using a GNN is able to reduce dramatically the calculation time of this kinds of really computer intensive calculations [11].

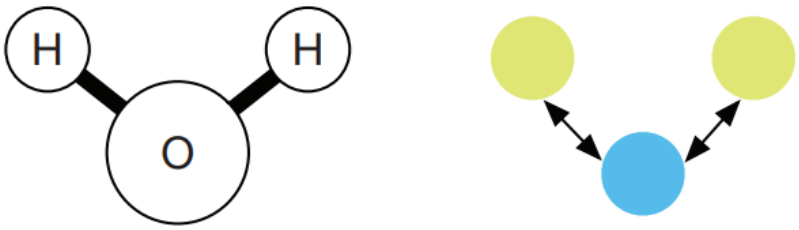


Figure 2: Molecules have a deep relationship between the atoms that compose them, and are a clear example of Graph with explicit relationship between nodes. [1]

Due to the explained deep relation between the different nodes that conform a graph, GNN take into consideration these relationships in the form of message passing m between the different connected nodes in a graph. The sequence of events that a GNN uses to work with the mentioned relationship between nodes is the following:

1. A node v_i in a graph will send a message m to another adjacent node v_i'
2. The node v_i' will then update its value using its current value, and the value from $m_i, i \in (1, 2, ..., adj(v_i'))$.
3. Once the update of the nodes in the graph has been performed, update the variable for the global attribute of the graph.

2.3 RouteNet characteristics

Our state of the art model differs from one standard GNN, as explained before a normal GNN updates the state of a node using the state of the adjacent nodes.

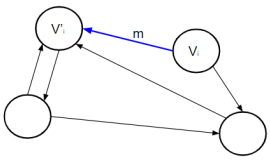


Figure 3: A message m is from v_i to v_i'.

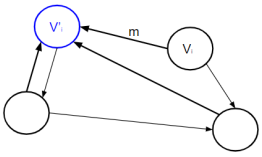


Figure 4: v_i' updates its state using the values received from adjacent nodes.

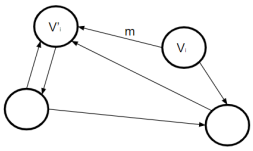


Figure 5: The Graph updates its state.

Unlike this approach, in our case what we need is a model that has two main characteristics, **paths** and **links**. The links are the edges between two nodes in a graph, the paths are the set of links that make up a routing between a source node and a destination node. As such our model is implemented with the idea that in order to understand the network model the links, which would be the edges in normal graph, become the nodes in said graph (the theoretical graph that the neural network creates), since the links are the ones that carry the delay information.

So, in order to know the state of a link h_{l_i} we need to know the state of the paths $h_{p_{1..n}}$ that use h_{l_i}, and in order to know the state of a path h_p it is needed to know the state of the links h_l that form that path. to sum up:

- h_p depends on the $h_{l_{1,\dots,k_{(|p_k|)}}}$ of the path.
- h_l depends on the h_p that contain that link.

These two principals can also be expressed in a more mathematical form:

$$h_{p_k} = g(h_{l_{k1}}, h_{l_{k2}}, ..., h_{l_{k(|pk|)}}) \tag{1}$$

$$h_{l_i} = f(h_{p_1}, ..., h_{p_j}), l_i \in p_k, k = 1, 2, ...j \tag{2}$$

Where f and g are a nonlinear system of equations, where the states are hidden variables, each function depends on the routing used, and the dimensionality of each function is is incredibly large.

Moreover the order on which the links are evaluated is essential in order to have a good performing model. If a link is lossy (produces drop packets) this will impact on the state of the coming links. As such a path is composed from a set of ordered links $p_k = (l_0, l_1, ..., l_n)$, and the delay of a path is indeed the sum of delays on every link of the path, $\sum_{i=1} d(l_{ki})$, where $d(l_{ki})$ is the sum of the delays on every link. The order from which the paths are calculated is not relevant.

2.4 RouteNet

The presented architecture is able to provide a routing invariant model, which at the same time is able to understand how a routing in a computer network behaves. RouteNet is able to overcome the problems raised by equations (1) and (2), by creating a GNN based on message passing between the links and paths, and updating the states of paths and links (h_p, h_l) using said message passing architecture. The algorithm below explains the architecture of RouteNet.

Data: x_p, x_l. R
Result: $h_p, h_l.y_p$
1 **for** $p \in R$ **do**
2 | $h_p = [x_p$,0, 0, ..., 0];
3 **end**
4 **for** $l \in N$ **do**
5 | $h_l = [x_l$,0, 0, ..., 0];
6 **end**
7 **for** *t from [0, 1, ..., T]* **do**
8 | **for** $p \in R$ **do**
9 | | **for** $l \in p$ **do**
10 | | | $h_p^t = RNN_t(h_p^t, h_l^t)$;
11 | | | $m_{p,l}^{t+1} = h_p^t$;
12 | | **end**
13 | | $h_p^t + 1 = h_p^t$;
14 | **end**
15 | **for** $l \in N$ **do**
16 | | $m_l^{t+1} = \sum_{p:l \in p} m_{p,l}^{t+1}$;
17 | | $h_l^{t+1} = U_t(h_l^t, m_l^{t+1})$;
18 | **end**
19 **end**
20 y = $F_{p(h_p)}$

The algorithm begins by initializing the states of both the paths and the links to 0, then proceeds to iterate T times, in order to achieve convergence, over the state vectors h_p, and h_l. This allows to reach a fixed point of a function from the initial states, which solves the problem of defining implicit functions. In order to overcome the second issue with routing invariance, the proposed model allows to perform message passing which unites topology and state vectors representation. This is done through the message passing solution, which are stated in lines 11 and 16, that allows links and paths to exchange information.

The message sent from the paths is the information of the current state h_p extracted by the Recurrent Neural Network (RNN), which used the current state of the path and the state of the links that the path uses (defined in line 9). The use of the RNN is because there exists a sequential dependence between the links on a path, which allows to propagate the information about the losses

in a path. While the message from the links is the sum of h_p that go through that link, we can use a simple summation in this case due to the order of the paths not being important, finally the state of the links for the next step is extracted from using another *RNN* from the current h_l and the received sum of the path messages, as stated in line 17.

Finally, the calculation of the predicted variable y_p is a function of the link and path features x_l, x_p, the predicted variable is extracted with a neural network.

In the figure below we can see a more graphical representation of the inner workings of the neural network.

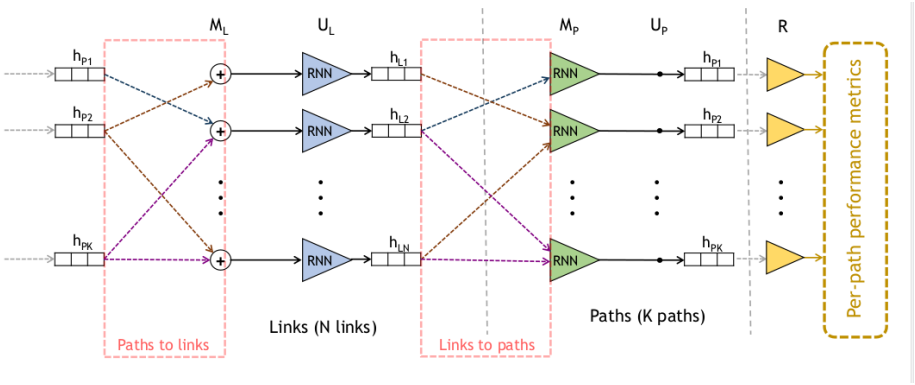


Figure 6: RouteNet Architecture.

In the figure it begins with the addition of the states h_p being added for each link that contains them, then the state of the link h_l is calculated using an *RNN*, with the value of h_l from the output of the *RNN*, it we proceed to calculate the h_p suing another *RNN*. The output of the h_p is directly the value of the message from that path.

RouteNet has already been successfully used to model one computer network [2], and recently to prove that it can generalize any computer network by training the model with multiple topologies.

3 Methodology

3.1 Introduction

The thesis tackles an standard problem of supervised machine learning [12] [13], meaning that we are on a problem in which a learning function (in this case RouteNet as a whole) maps the input features to an output based on a set of examples input-output pairs. A supervised learning algorithm is able to infer a function from the analysis of the training dataset which can be used to predict the output of unseen data in a reasonable way, the assumptions that the algorithm takes in order to predict the output of unseen examples is also know as *Inductive bias* [14]

We are also using a hold-out test set, which means that we are splitting our training dataset between train and evaluation (test). The split is done in 80% - 20% capacity and using an stratified sampling approach. The variable used to stratify is not the target attribute as it usually is, but instead is a parameter called λ which I will explain later on.

3.2 Dataset

3.2.1 Data Generation - Simulations

In order to obtain the dataset to train with RouteNet it is first needed to simulate a real network environment, to do so the simulation tool Omnet++ [15] is used to simulate networks and produce the desired dataset. Each simulation is done in a given routing by four different traffic intensities λ, each simulation is repeated 500 times in order to achieve a proper mean value for the delays in each path. In our case all nodes are connected to all the others nodes in the network with different routings. More over each simulation (meaning one sample) used to take 16050 seconds. As you can imagine this meant that the simulations took a long time to complete, to the point where it was impossible to finish the creation of a proper dataset due to the time needed exceeding 1 year.

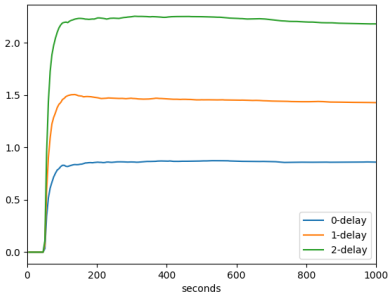


Figure 7: Evolution of delay over time for 50 nodes for each traffic intensity λ.

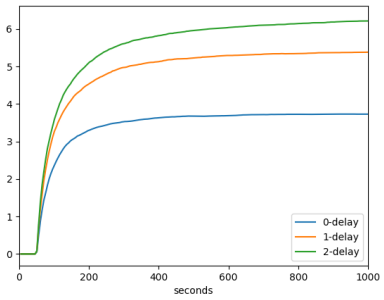


Figure 8: Evolution of delay over time for 100 nodes for each traffic intensity λ.

It became quite clear for us (myself and the research team) that we were simulating way more time than what was necessary in order to achieve an stable value of a given delay, as such it was decided to cut the simulation time in order to achieve a more reasonable time for the simulations.

Once the simulation time was reduced to a reasonable amount of time, around 400 seconds per sample, to perform correct simulations, we were able to get the necessary data to create the model.

3.2.2 Dataset

In order to create a sizable data sample simulations were performed on three different network graphs, with different sizes, each with 14, 24, and 50 nodes. In the figures below we have the different topologies.

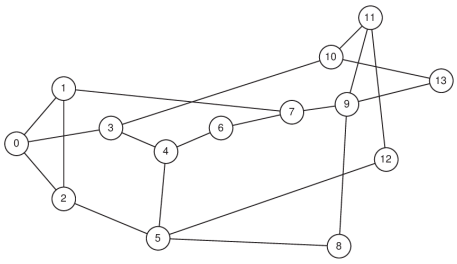


Figure 9: Network graph with 14 nodes.

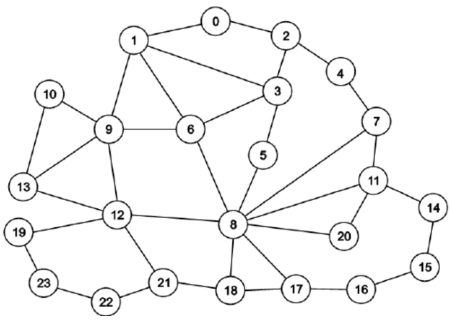


Figure 10: Network graph with 24 nodes.

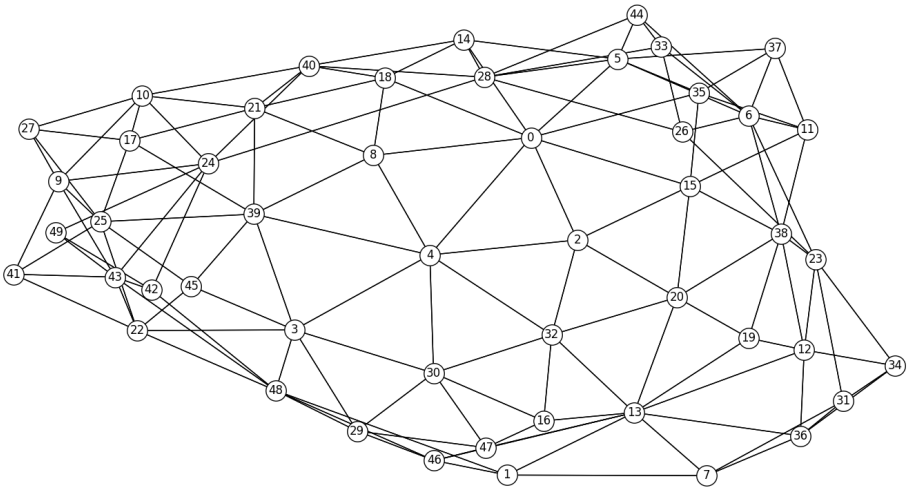


Figure 11: Network graph with 50 nodes.

These three topologies are synthetic (meaning that they do not come from real networks but instead are computer made) [16] [17] [18], but the 14 nodes and 24 nodes topology are based on real networks. For each topology shortest path routings algorithms where performed for every pair of nodes, meaning that on e given node in a network graph of N has a total of $N-1$ routings, since no self loops are allowed. The reason why no self loops are allowed is because in a network environment they make no sense. This means that a network topology has a total of $N*(N-1)$ routings in total. In the following table we can see how many routings have each of the network topologies that we are using in this research. This means that we in theory have 3 different datasets, one for each topology type.

	14 Nodes	**24 Nodes**	**50 Nodes**
Routings	182	552	2450

Table 1: Number of routings per topology

3.3 Routing Creation

A central point of the thesis is the ability to create an extensive amount of routings which can be used as a representative sample for training RouteNet. Since RouteNet should not care about the network size, the creation of a multitude of routings from which the nodes of the GNN can infer a function to predict unseen examples is of great importance. In the end, RouteNet will try to predict the delay of a path by using the data from the nodes which it has been trained with,

this means that the real important factor is that the nodes from the training dataset have as many possible combinations of routings going through them as possible.

In order to perform the routing creation algorithm a *Deep-first search* strategy, which was presented by *Charles Pierre Trémaux* as an strategy for solving mazes, has been implemented.
The version of the algorithm used is the one explained in the book Introduction to algorithms [19], which establishes a cutoff value in order to keep the algorithm to a reasonable time performance. The idea is to use the algorithm to search all the possible paths shorter than the cutoff value, from one source to a destination. The cutoff value has always a base value of the shortest path distance between the two nodes The algorithm works in the following manner:

```
   Data: The topology as G, s as a vertex of origin, and d as the
         destination, cut as the cutoff value
   Result: Routings for source - destination pairs
 1 Function DSF(G, s, d, cut) is
 2 |   let S be a stack of all the adjacent nodes to S;
 3 |   let V be the visited nodes;
 4 |   while S is not empty do
 5 |   |   v = S.pop();
 6 |   |   neighbour = Adjacent nodes of v
 7 |   |   if c is empty then
 8 |   |   |   S.pop();
 9 |   |   |   V.pop();
10 |   |   end
11 |   |   else if length V ¡ cut then
12 |   |   |   if neighbour is the d then
13 |   |   |   |   return V + d;
14 |   |   |   end
15 |   |   |   else if neighbour not in V then
16 |   |   |   |   V.push(neighbour);
17 |   |   |   |   S.push(Adjacent nodes of neighbour);
18 |   |   |   end
19 |   |   end
20 |   |   else cutoff value reached
21 |   |   |   S.pop();
22 |   |   |   V.pop();
23 |   |   end
24 |   end
25 end
```

This algorithm is able to generate a full valid routing for the whole network topology (in this case the 50 nodes topology) in under 1.5 seconds, which means that we are able to generate quite a big amount of routings in a short span

of time. The procedure described above is repeated 10 times for each source - destination pair, in order to have a sizable pool of different paths for each pair of nodes.

When I refer to a valid routing it means a routing that does not have loops in the routing procedure. It is important to understand that in networking routing policy is usually destination only, meaning that the source node does not have any impact on the route to use, in other words, when a IP packet arrives on a network node, it only takes into consideration the destination of the packet and forwards it to the node port specified in the IP table. This can lead to loops in the network, for example:

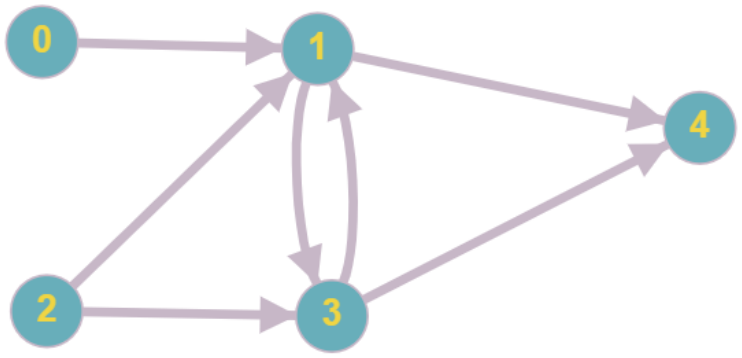


Figure 12: Example of routing with loops.

Here we have an small topology consisting of 5 nodes, lets say we have 2 routing, one having the source node as node 0, and destination node as node 4, and the other routing with source 2 and destination 4. The path for each routing are:

$$R1 : Node0 -> Node1 -> Node3 -> Node4$$

$$R2 : Node2 -> Node3 -> Node1 -> Node4$$

In this case we have a loop, since the nodes only care for the destination of the packet, in *R1* once the packets gets to node 1, it will be sent to node 3, but due to *R2* establishing that packets that arrive to Node 3 with destination Node 4 have to go to Node 1, the packets will be sent once again to Node 1, which in turn will send it once again to Node 3, thus creating a loop.

Moreover the algorithm take a parameter which we will call α, that indicates the algorithm how it should attempt to place the paths on the graph. At the moment the α parameter takes three possible values, *easy*, *normal*, and *hard*.

	Routing placement
easy	Sparse placement
normal	Random placement
hard	Overlapping placement

The algorithm works by assigning weights on each link, depending on the α chosen the algorithm will choose paths that contain the links with less, more, or random weight. This allows us to generate a representative sample of routing scheme, with different values of delay for each path. The aim is to create enough routings of different length to have a representative sample of paths, which can be used to train a model to generalize any network.

The figures below show the routing placement for each algorithm, the red lines indicate the routing placements, the thicker the line, the more routings go through that link.

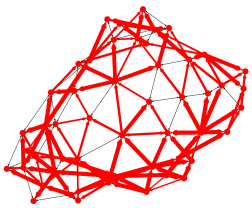


Figure 13: Random placement of routings.

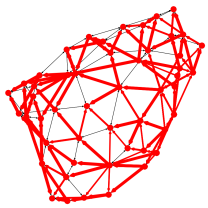


Figure 14: Sparse placement of routings

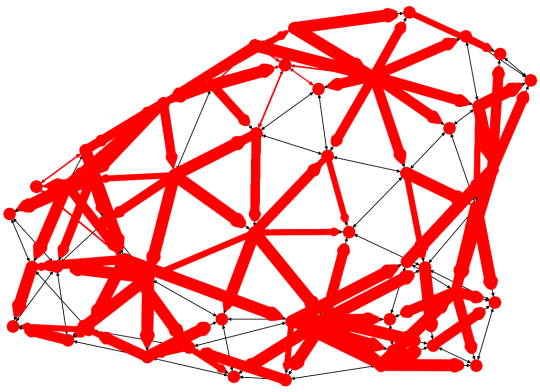


Figure 15: Overlapping placement of routings, we can see how the thickness of the red lines is much bigger than for the other two figures.

In the case of the *hard* placement the average weight of the links used is around 144, while in the case of the *easy* placement, is around 116. This means that indeed the *hard* placement method uses a single link more times than the easy placement. The idea is that the algorithm aims to explore as much as possible the routing schemes possibilities that the topology offers.

I want to remark that it is nearly impossible to fully explore a network topology the size of 50 nodes, since the number of possible combinations is in practical terms infinite. As such even if the algorithm is able to generate less than 1% of the total possibilities the generated routing schemes would probably be enough.

With this algorithm we are able to generate a multitude of routing schemes that explore the topology in a bigger number of ways than an standard shortest path algorithm would, thus creating the basis for the main objective of this thesis, the theory is that in order to be able to predict the delay of any model the number of nodes on the topology is irrelevant, but instead the data should be a sample representative enough of routings.

Due to time constrains, we were not able to implement the proper simulation policy, which works with a source-destination model, for implementing the routing algorithm. This causes that the algorithm is only able to create combinations of shortest path, which is not ideal, but is enough to create an small representative sample.

The algorithm has been implemented using the *Python* programming language [20] with the help of the *networkx* package [21]. This package offers methods to ease the working with graphs data structures.

Once the routing have been created, each is simulated for different values of traffic intensities, called λ, these λ represent the amount of traffic that is produced by each source destination pair. In Table 2 .

λ	**8**	**9**	**10**	**12**	**15**	**Total**
14 Nodes	156		174	91	174	595 aprox 3.8GB
24 Nodes	174		170	172	174	693 aprox 13.3GB
50 Nodes		200		200	200	600 aprox 42.8GB
50 Nodes Alternative		139		138	138	414 aprox 36.3 GB

Table 2: Simulations per lambda and nodes

The difference between *50 Nodes* and *50 Nodes Alternative* are the way the routing are created. The routing for 14, 24, and 50 Nodes are standard Shortest Path routing, while the ones created for *50 Nodes Alternative* are the ones we aim to prove our theory in the thesis, which is creating a representative sample enough of routings in order to be able to predict any other routing regardless of the topology size. Moreover the 50 Nodes dataset has an scaling factor applied on it which will be explained later on.

Each combination of routing per λ is then simulated 500 times for the 14, 24, and 50 Nodes, and 300 times for the 50 Nodes alternative. In the end the total amount of samples for each topology is the the total number of routings, multiplied by the total number of λ, multiplied by the total amount of simulations performed for each combination of routing and λ.

	14 Nodes	24 Nodes	50 Nodes	50 Nodes Alternative
Samples	297.500	346.500	300.000	124.200

Table 3: Total amount of samples per each routing.

3.4 Training and Evaluation

As explained in the methodology the split between training and evaluation is done through and stratified sampling using the λ as the parameter to create the subpopulations from. The split for train and evaluation is 80% and 20 %.

The Model is **only** trained using the 50 Nodes Alternative data. The other datasets are used as evaluation only. The table below shows the split in sample numbers for training and evaluation.

	Training	Evaluation
Samples	99.360	24.840

Table 4: Training and evaluation split.

All the data is normalized during the pre processing phase, the normalization of the target feature (delay) is done by calculating the mean and the standard deviation of a representative sample of the data and proceeding to calculate the standard score for each sample.

3.5 Data Features and Statistics

The dataset is composed of quite a large number of features. A single sample is a whole topology with all the delays for each source destination pair routing, meaning that, for example, a Routing with $N = 50$, has a total amount of $49*50$ *delays*, *bandwidth*, *jitter*, *links*, *Number of packets*, *drops*..

- **Delays**: Target feature. It's a continuous variable containing the delay of each path in the network.
- **Bandwidth**: Also known as a traffic matrix, a continuous variable which is the set of bandwidth that each path has.
- **Jitter**: Variation of the delay. Can also be the target variable, in the case it is not the target, it is not used.
- **Drops**: Continuous variable. Number of packets lost for each source destination combination.
- **Number of packets**: Continuous variable. Number of packets sent between source and destination.
- **Links**: Number of hops that the path takes from source to destination. (Calculated at runtime)

3.5.1 50 Nodes Alternative

The figures below indicate some of the statistics of the dataset for 50 Nodes Alternative, which is the dataset used for the training of the model. The figures do not use all the data in the dataset, but only use 1800 samples due to size constrains of the dataset. Figures 16, 17, 18, 19 show the distribution of values for the bandwidth, delay, packets, and drop packets.

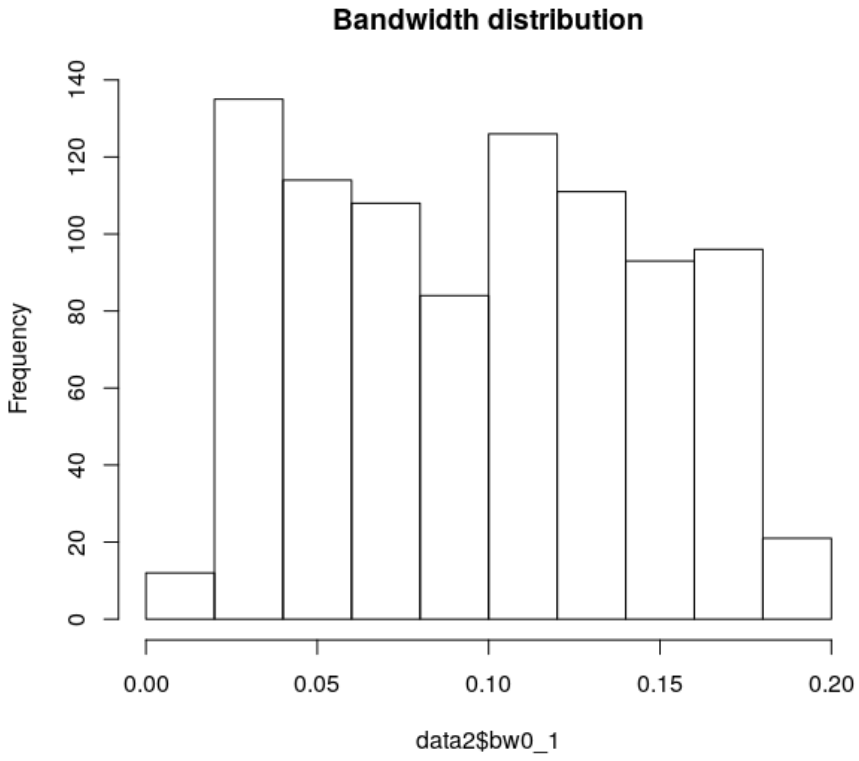


Figure 16: Histogram of bandwidth distribution

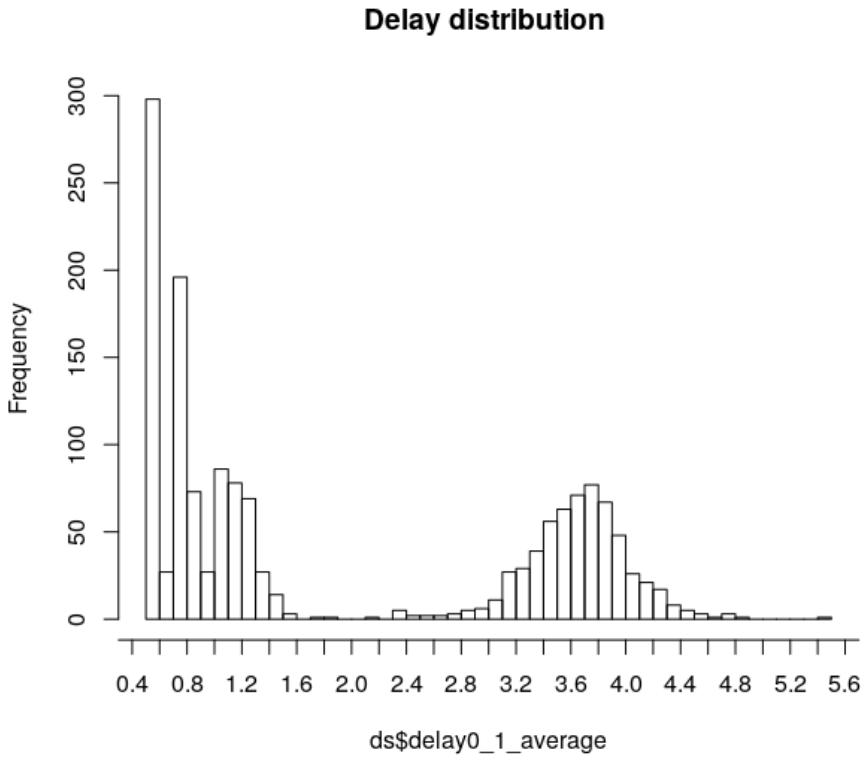


Figure 17: Histogram of delay distribution

This is an important figure, delay is indeed the target variable, so the distribution of values that our dataset has is quite important in order to know which range of values RouteNet is able to predict, values outside the range shown in the histogram will be hard to predict for the model.

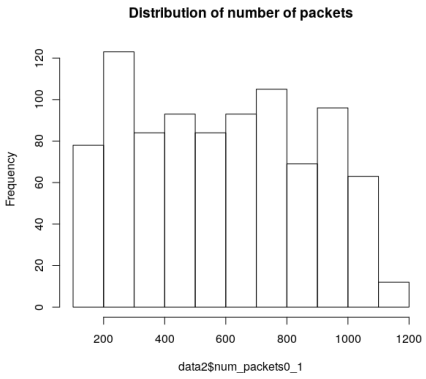


Figure 18: Histogram of packets distribution

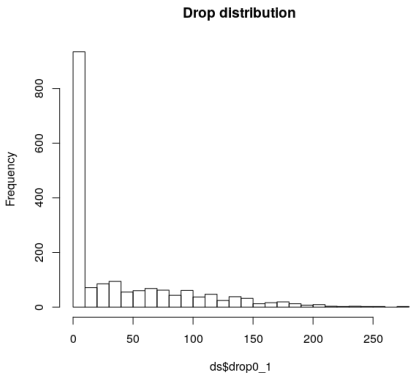


Figure 19: Histogram of drops distribution

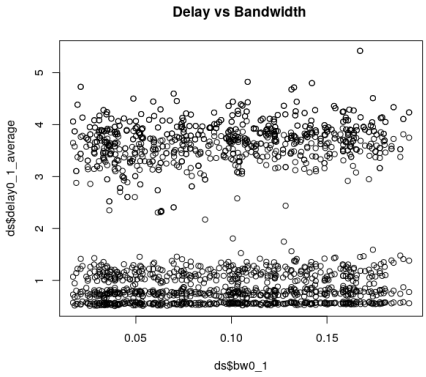


Figure 20: Delay vs Bandwidth

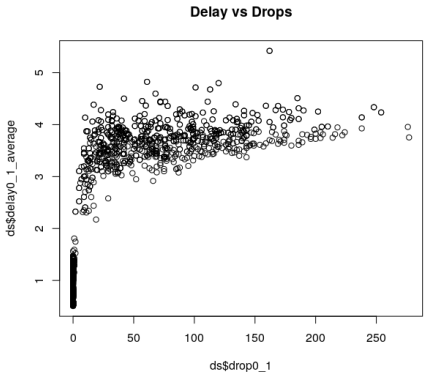


Figure 21: Delay vs Drops

Figure 19 indicates that we are losing a few packets during the network transmissions. Figure 20 shows the relation between the delay and the bandwidth, we can see how the delay is not affected much as the bandwidth increases. If we look at Figure 21, we can see how the delay is indeed affected by the number of drops in the network. The data seems to show that the delay vs drops packages follows a logarithmic function.

The Table below 5 shows the pearson correlation between each feature. We can see how the bandwidth, number of packets, and dropped packets all have some degree of correlation to the delay. And as expected by looking at the figures above, the dropped packets is the most correlated feature to the delay. There are datasets where no dropped packets exist, in those cases the bandwidth is the most correlated feature to the delay.

	BW	# Packets	Drops	Delay
BW	1.0000000	0.9982516	0.6240850	0.1521073
# Packets	0.9982516	1.0000000	0.6251060	0.1488149
Drops	0.6240850	0.6251060	1.0000000	0.3540376
Delay	0.1521073	0.1488149	0.3540376	1.0000000

Table 5: Correlation between features for the 50 Nodes Alternative dataset

Finally the figure below shows some statistics for the dataset.

```
    bw0_1          num_packets0_1      drop0_1         delay0_1_average
Min.   :0.01713   Min.   : 100.0   Min.   :  0.00   Min.   :0.5067
1st Qu.:0.05292   1st Qu.: 321.0   1st Qu.:  0.00   1st Qu.:0.7300
Median :0.09964   Median : 596.5   Median :  0.00   Median :1.1557
Mean   :0.09744   Mean   : 585.8   Mean   : 31.98   Mean   :1.9595
3rd Qu.:0.13815   3rd Qu.: 840.2   3rd Qu.: 51.00   3rd Qu.:3.5673
Max.   :0.19293   Max.   :1121.0   Max.   :277.00   Max.   :5.4211
```

Figure 22: Dataset Summary for 50 Nodes Alternative

3.5.2 50 Nodes

Although the dataset from 50 Nodes and 50 Nodes alternative might seem similar at first glance, the reality is that the only thing that both datasets share is the same topology. The difference between the two is that the data for the 50 Nodes dataset has been scaled by a factor that takes into consideration the size of the network. This procedure reduces drastically the traffic in the network, but it also produces no packets dropped, which ends up with less delay on the network, but also with a lower value of bandwidth. This factor has been applied due to the 50 Nodes Alternative dataset having some high values of delay which are not considered acceptable from a networking point of view. Thus this correction factor allows to produce traffic which creates acceptable levels of delay.

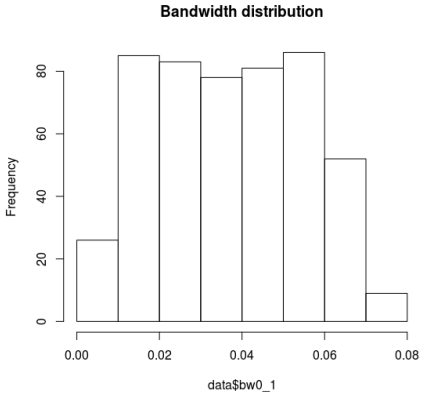


Figure 23: Distribution of the bandwidth for dataset with 50 nodes.

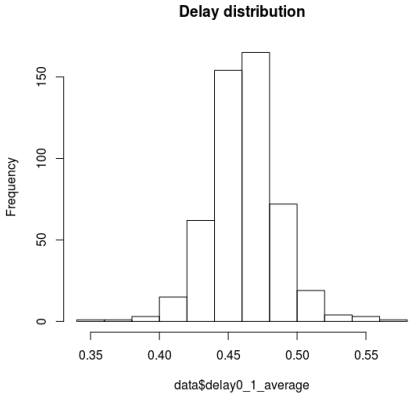


Figure 24: Distribution of the delay for dataset with 50 nodes

If we compare the bandwidth histogram for 50 Nodes Alternative dataset in Figure 16 and the one for the current dataset in Figure 23 we can clearly see how the bandwidth for the current dataset is approximately a magnitude 10 times smaller than the one for the 50 Nodes Alternative dataset.

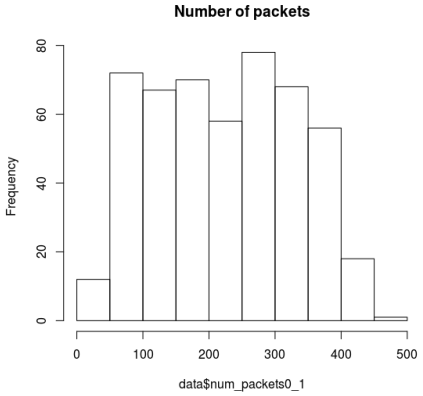


Figure 25: Distribution of the packets for dataset with 50 nodes.

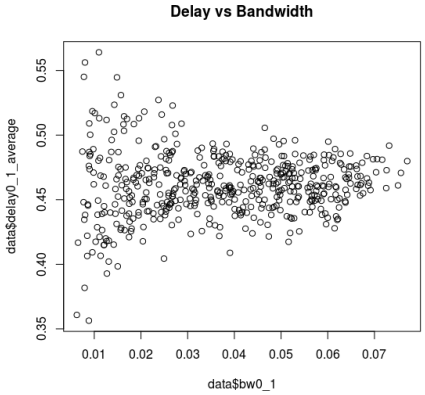


Figure 26: Delay vs Bandwidth for 50 Nodes

If we compare again the number of packets in the network Figures 18 25 it is also clear how there are much more packets in the dataset 50 Nodes Alternative than the 50 Nodes dataset. This clearly shows that there is less traffic in the network.

3.5.3 24 Nodes

The below figures are the statistics for the topology with 24 nodes, as with the data from the 50 nodes alternative dataset, we are not using all the data samples, but are only part of it. Also, the dataset of the topology for 24 nodes does not contain the same information as the other topologies due to a change in the data format that occurred at the end of the thesis period, the main difference is that there is not attribute for the total amount of packets from source - destination.

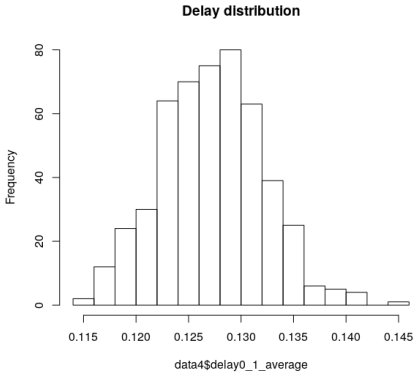


Figure 27: Distribution of the delay for dataset with 24 nodes.

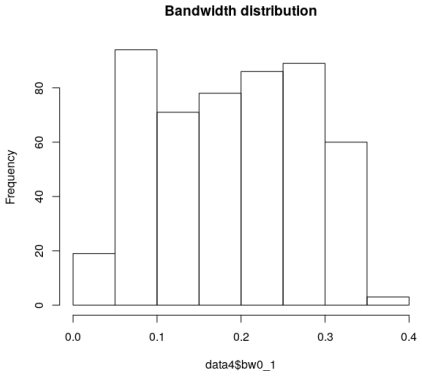


Figure 28: Distribution of the bandwidth for dataset with 24 nodes

The main difference that we find in this subpart of the dataset compared to the figures from the training data is the values that the delay achieves, if we look at Figure 17 and compare it to Figure 27, it is clear that the topology of 24 nodes achieves lower rates of delay compared to the ones from the 50 Nodes. This is believed to be a cause of more traffic going through each link in the topology of 50 nodes, since we have way more routings, also, the fact that more hops are needed due to the diameter of the graph being higher is also a cause of higher delay in the network.

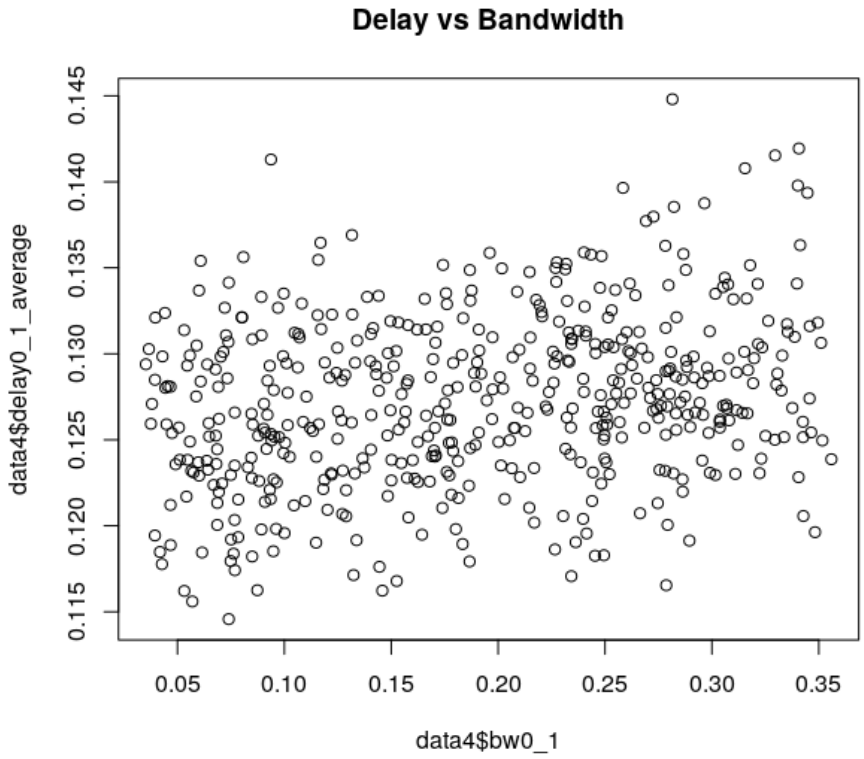


Figure 29: Delay vs Bandwidth for dataset with 24 nodes.

With the delay vs bandwidth plot there seems to be no clear relation between the two, but it seems like the delay increases a bit as the bandwidth goes up. The drop vs bandwidth plot is not shown due to there not being any packets drops in the computer network.

3.5.4 14 Nodes

Finally the topology with 14 Nodes, due to this being the topology with the least amount of nodes, it is also the topology with the lower amount of delay in the system. The delay vs bandwidth 31 also seems to indicate an small increase of the delay when the bandwidth increases. Figure 30 also indicates how the delay spikes when the dropped packets increase.

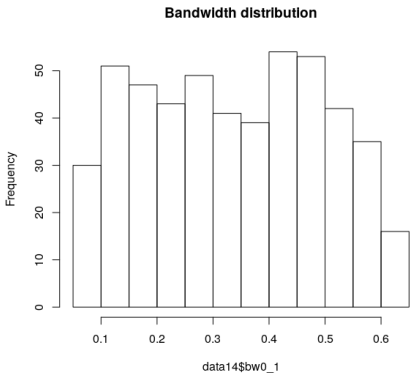


Figure 30: Distribution of the bandwidth for dataset with 14 nodes.

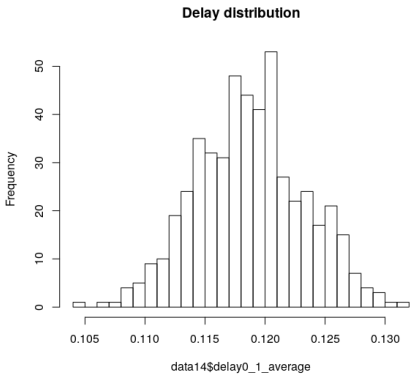


Figure 31: Distribution of the delay for dataset with 14 nodes

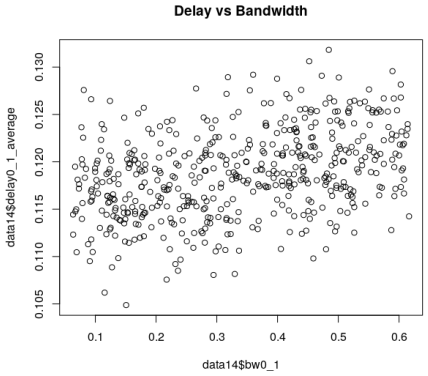


Figure 32: Delay vs Bandwidth for dataset with 14 nodes.

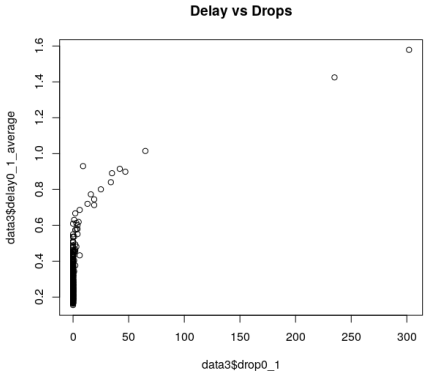


Figure 33: Delay vs Drop for dataset with 14 nodes in cases where there are dropped packets.

4 Technical Details

4.1 Introduction

In the dataset section I will go through how the data is gathered, simulations are performed, which different techniques have been used to generate the data, an statistical analysis of the datasets, and how this data generation impacts the model.

4.2 Architecture

RouteNet is implemented using the Tensorflow and Keras library. Tensorflow is, as described by Tensorflow creators, "An end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications" [22].

Tensorflow eases the creation of machine learning models, by providing efficient and custom model creation, implements performance metrics, and provides with an easy to use tool to check the results of the models created in the form of *tensorboard* [23].

Keras is a high level neural network API which is capable of running of top of Tensorflow [24]. Keras is used due to the more friendly interface for coding than Tensorflow.

4.2.1 Gated Recurrent Unit

The RNN are used one for the path and the other for the links, the RNN gating mechanism chosen for this implementation are Gated Recurrent Units (**GRU**) [25], the use of GRU over other mechanisms such as Long Short Term Memory (LSTM), is mainly due to achieving similar performance while using less parameters and thus being simpler than an LSTM gate [26] [27]. The need for using these kinds of gates is due the whats called *vanishing gradients* problem, this occurs when a gradient shrinks as the back propagation mechanism proceeds, if a gradient value becomes small after some training, the model stops learning, and due to how the back propagation mechanism works, small gradients usually appear in the beginning layers of the Neural Network, thus, stopping all learning of the model. Figure 34 shows an example of vanishing gradients.

Figure 34: Gradient vanishing problem

In the Figure below (35) we can see the inner structure of a GRU gate.

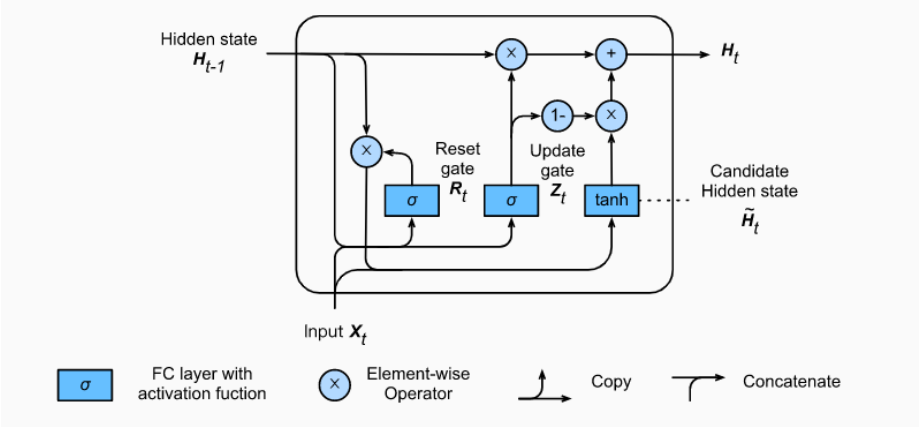


Figure 35: Gated Recurrent Unit [28]

Both the reset gate and the update reset gates are both *sigmoid* functions, which squish the value that go through it between 0 and 1, this allows the neural network to keep or forget data, a value which is converted to 0 is forgotten, while the others value become more or less relevant as time goes on. This mechanism allows the gate to keep or forget important and unimportant data. The *tanh* activation function is used to regulate the flow of data in the network, the output of the *tanh* function are the values squished between -1 and 1.

A quick summary of what each gates does, the update gate helps the network how much of the past information needs to be passed along to the next steps. Similarly the reset gate is used how much of the past information will be forgotten.

4.2.2 Readout function

The readout function is an standard neural network with two hidden layers which uses a *Scaled Exponential Linear Units* or *SELU* for short, the main characteristic of SELUs are the internal normalization [29] and also solving the vanishing gradients problem. The readout neural network also uses an **L1** regularization, also know as Lasso regularization, which is useful not only as a regularization procedure in order to reduce the weights of the features that are not important, but also as a feature selection procedure in case there is a feature that is not interesting for the model. Choosing an L1 regularization is due to mostly not performing a feature selection beforehand, and as such it is performed implicitly on the model [30].

The readout functions are mixed with two dropout layers, in the training phase, they prevent overfitting, while during the inference, they are used for Bayesian posterior approximation [31]. This is an important feature of the model as it allows us to asses the confidence of network prediction, what this means is that when a Neural Network is trained and optimized for an specific output, the solution of the optimization can be for which a network is too optimistic, meaning that a normal neural network without the dropout layers does not take into consideration the uncertainty that another Bayesian model could achieve, which is difficult to implement due to the infeasibility due to the computational power required to train such model.

During the training phase half of the neurons of a given layer will be deactivated, forcing the layer to learn with different neurons the same concept.

Quoting [31] "Using a dropout layer allows to extract the necessary information from the model that has been thrown away so far. This mitigates the problem of representing uncertainty in deep learning without sacrificing either computational complexity or test accuracy."

Repeating inference multiple times with dropout layers being active gives a distribution of results. The spread of this distribution is a measure of network confidence about the prediction and the optimal point.

Over all, the use of a dropout layer allows the neural network to trade training performance for better generalization of the model.

An other important item to choose is the algorithm used in order to minimize the loss function. The loss function is the metric that is going to be used as the performance metric for the model to know if the back-propagation algorithm is working or not, meaning that if the loss function increases, the model is not working properly, but if instead decreases, it means that the weights computed by the back-propagation algorithm are going in the right direction.

4.2.3 Loss minimization function

The algorithm used in order to minimize the loss is the *Adam Optimizer* [32]. Adam is defined by its creator as a combination of both RMSprop and Stochastic Gradient Descent with momentum, both of which are also used as optimizer for neural networks. The attractiveness for using Adam Optimizer are the following [33]:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

All of these characteristics are the reasons why Adam Optimizer was chosen over other kinds of optimizers. Adam Optimizers maintains a learning rate that improves performance on problems with sparse gradients and also are adapted based on the average of recent magnitudes of the gradients for the weight.

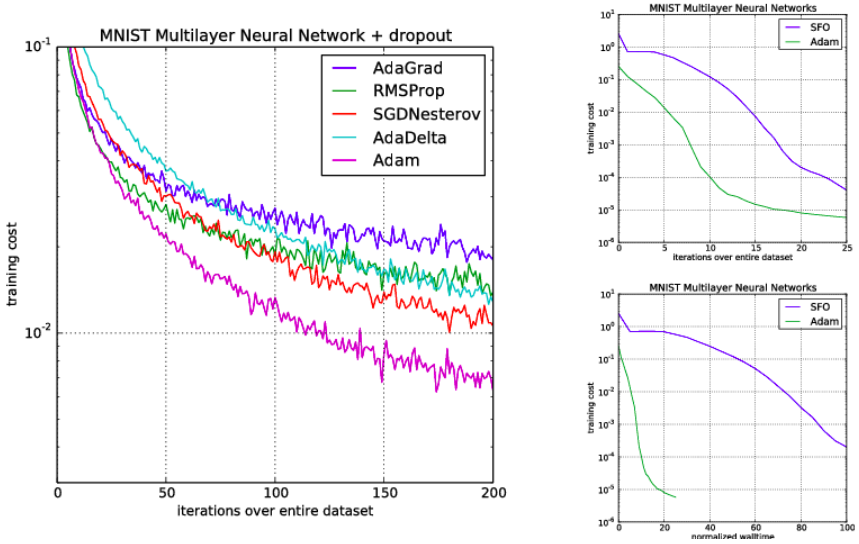


Figure 36: Adam performance compared to other models as described in [32], the lower the better.

4.2.4 Performance Metrics

With the model created it is needed to define some performance metrics, not only to use during the evaluation process but to also define which should be the metric that the algorithm should try to optimize, also called the loss, and to choose which method should be used in order to optimize it.

The metric implemented are:

- Mean Absolute Error: which measures the difference between two continuous variables.
$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y_j}|$$
- ρ: which is a Pearson correlation, which indicates the extent to which two linear variables are correlated.
$$\rho = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$
n is the sample size, x_i, y_i are the samples, $\bar{x}$ is the mean
- Mean Relative Error: which is a measure of the uncertainty of measurement compared to the size of the measurement.
$$MRE = \frac{1}{n}\sum_{j=1}^{n}\frac{y_j - \hat{y_j}}{y_j}$$
- R^2: which is the standard metric to evaluate the goodness of a model.
$$R^2 = 1 - \frac{\sum_i(y_i - \hat{y_i})^2}{\sum_i(y_i - \bar{y})^2}$$

In our model we have chosen the Mean Absolute Error as the *loss* metric, meaning that this is the metric that RouteNet Adam optimizer will try to minimize, the other metrics will be used as evaluation.
Besides the metrics mentioned there will also be used as metrics for the goodness of the fit the True delay vs predicted delay scatter plot, one per each routing, it also contains the distribution of points as an histogram. There will also the cumulative distribution function of the mean relative error.

5 Evaluation

5.1 Introduction

In this section I will explain the results of our experiments and check if the hypothesis stated in the objectives of this thesis are achieved. We will do this by showing the results of the metrics explained in the previous section. I will also evaluate the performance of the model.

5.2 Metrics

5.2.1 50 Nodes Alternative

In this section I will discus the metrics results for our model. I will begin by showing the evolution of the different metrics for the training and evaluation dataset over time, commenting the results. Finally I will show the results for the other topologies.
Figure 37 shows the evolution of the loss (Mean squared error) metric, which is the metric that the model tries to minimize.

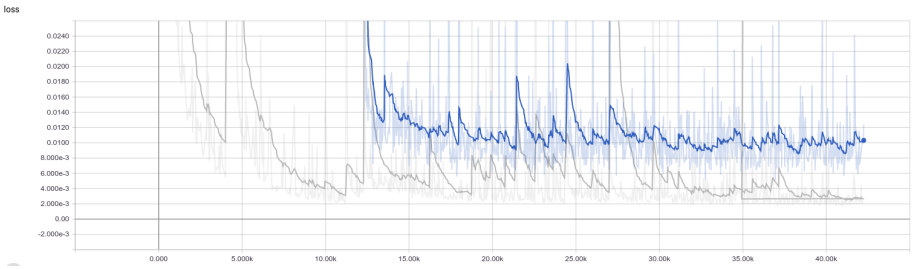


Figure 37: Loss (MSE) for training and evaluation data. The blue line indicates the evolution of the training data, the grey line indicates the evolution of the evaluation data.

Although it may seem strange that the validation error is lower than the training error, this is due to the use of the dropout regularization layer in the neural network and the L1 weight regularization mechanism. In the Keras documentation (which is the library used to create the neural network) its stated "A Keras model has two modes: training and testing. Regularization mechanisms, such as Dropout and L1/L2 weight regularization, are turned off at testing time." [34]. This makes it so the evaluation function is more smooth than the training, and thus providing better results.

The training line starts later than the evolution due to tensorflow only storing the past 30K steps for training. The loss for the training has a final value of ***0.01063***.

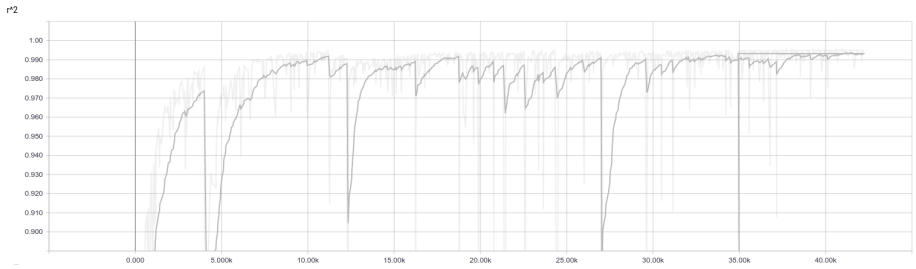


Figure 38: R_2 for training and evaluation data.

The drops seen in the evolution are due to the training not being done all at once, but instead training it for a number of steps, and then restarting the training. The following plots show the evolution of the performance metrics for the evaluation dataset. All the results below are for the part of the evaluation data for the dataset called 50 Nodes Alternative.

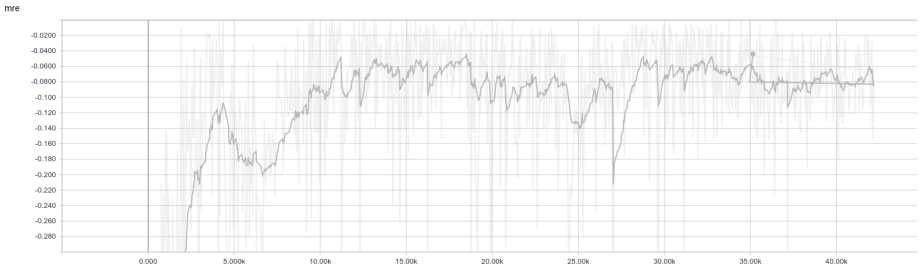


Figure 39: MRE for training and evaluation data.

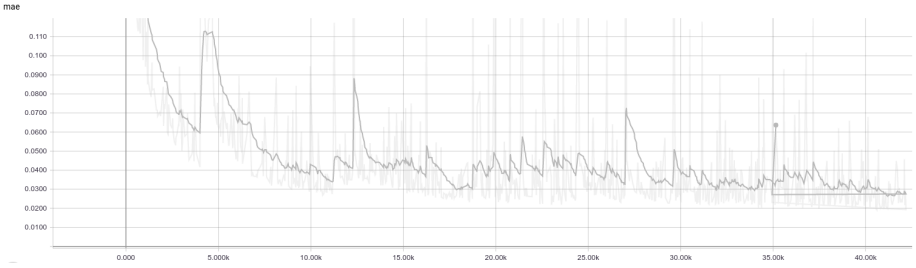


Figure 40: MAE for training and evaluation data.

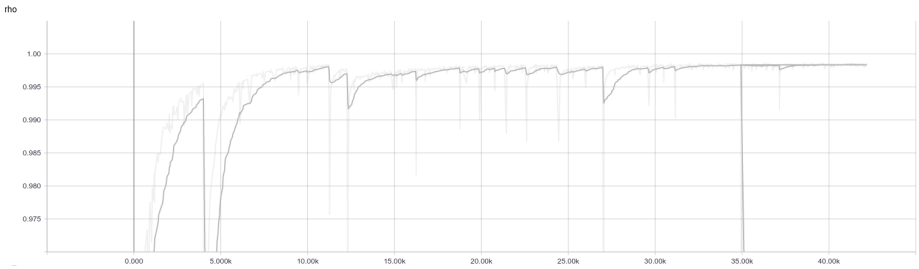


Figure 41: ρ for training and evaluation data.

	R^2	ρ	**MAE**	**MRE**	**MSE**
Evaluation	0.9944	0.9985	0.02685	-0.0786	0.00289

Table 6: Table of results for evaluation dataset

It is clear that we are achieving a really good model for our first dataset, 50 Nodes Alternative, all metric indicate a low error, the ρ metric shows that there is indeed a great linear correlation between the predicted delay values and the true delay. The model also has a really high R^2 value, showing that our model explains a great amount of variation of the response variable (delay) is explained by our model. Finally RouteNet achieves a low amount of error of the predictions (mre, mae, mse).

Finally we can use a simple plot of True delay vs Predicted delay for a more visual representation of the accuracy of the results and the model. The histograms on the side of the plots indicate the distribution of the samples.

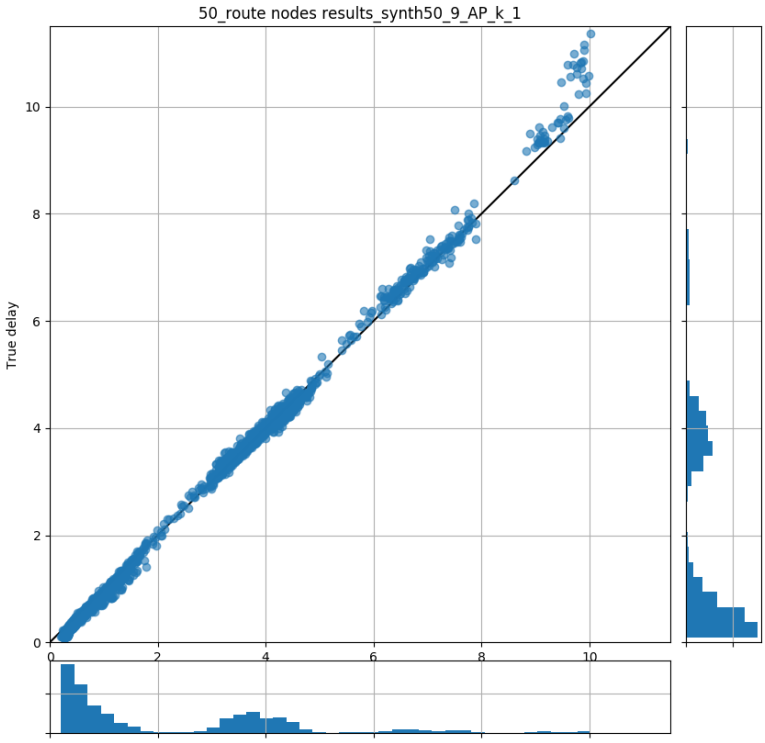


Figure 42: True delay vs Predicted for 50 nodes Alternative.

The model predicts nearly perfectly the values of the delays, the only exception seems to be for really high values of delay, which are not realistic in a computer network.
More True delay vs Predicted scatter plots can be seen in the appendix 8,
All these results above are for the dataset for 50 Nodes Alternative, below I will show the results for the rest of the datasets.

5.2.2 50 Nodes

The dataset for 50 Nodes is similar to the one used for training, but as explained before, there is less traffic in the network, this leads to lower values of delay but also way lower values of bandwidth.

	R^2	ρ	**MSE**	**MRE**	**MAE**
50 Nodes	0.86	0.96	0.00414	-0.0377	0.047

Table 7: Metric results for 50 Nodes.

The metrics provide good results, we achieve a lower value of R^2 than on the original dataset, but that to be expected. The error values are also quite low, which si another indicator of the good performance of the model.

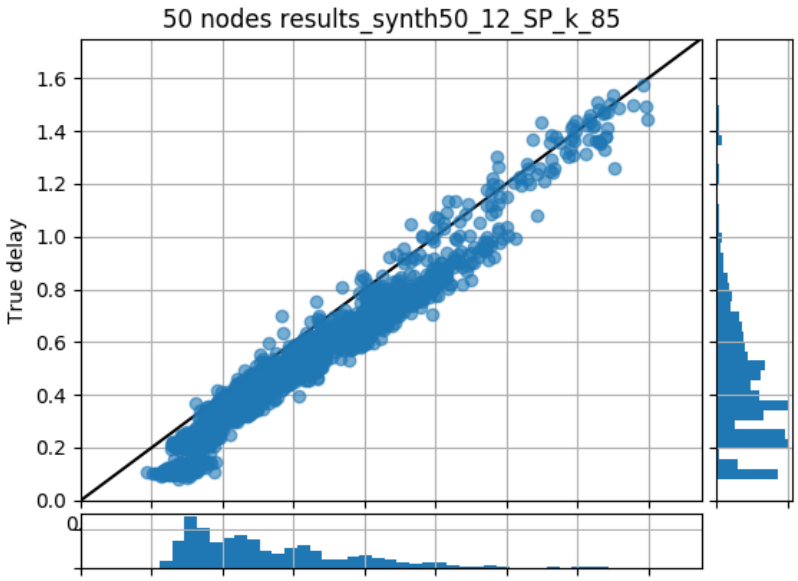


Figure 43: True delay vs Predicted for 50 nodes.

The model seems to predict delays to be somewhat higher than their actual value. Yet, overall the results seem quite good, there are no values which seem far off of the regression line. It appears that RouteNet tends to predict values with higher delay value than the real delay. We can say that the model is able to predict correctly the values for this particular topology.

5.2.3 24 Nodes

The 24 Nodes topology is the first dataset which has a different topology model than the training dataset, so this is the first test to check if our hypothesis stated in the objectives, which says that RouteNet trained with routing fro only one topology is able to model any other network independently of the size, holds or not.

	R_2	ρ	**MAE**	**MRE**	**MSE**
Evaluation	0.946	0.9861	0.21	-0.115	0.34

Table 8: Table of results for 24 nodes.

If we take a look a the metrics above, the results are really good, especially for the R^2 and ρ, which indicates that the model is indeed able to predict the delay values for topologies of different size, and obviously, of different form. The error values are indeed higher than the ones from the 50 nodes, which is indeed concerning, but the R^2 and the ρ values are really high, which indicates a good performing model overall.

The error being high can usually be solved by adding more training data, so in future experiments this should be taken into consideration. I do also believe that the high error value, especially the *MAE*, comes from not being able to correctly predict high delay values correctly, and thus increasing the overall error.

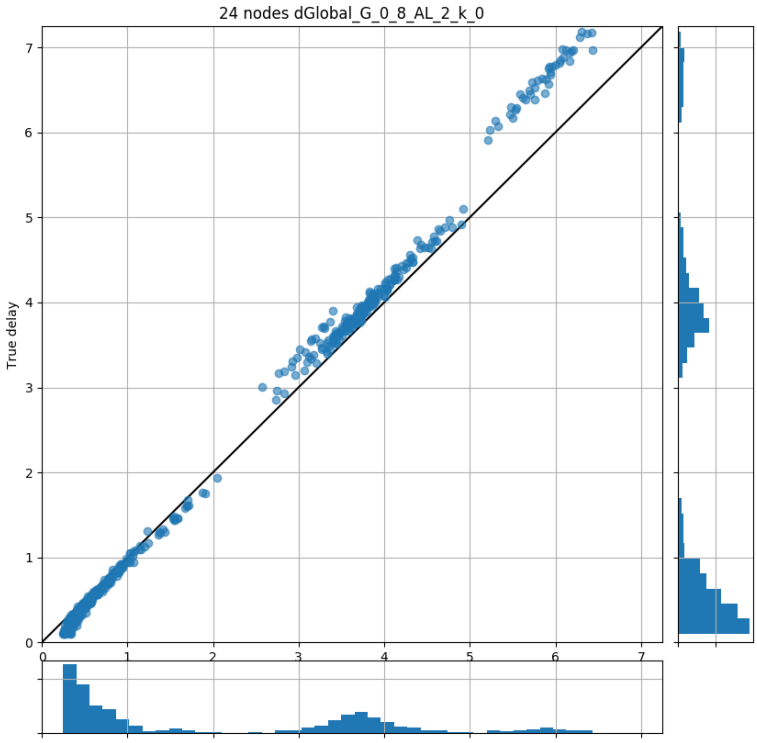


Figure 44: True delay vs Predicted for 24 nodes.

Yet again, we see how the model correctly predicts the values of delay, except for the really high values, which once again, are not realistic.

5.2.4 14 Nodes

The 14 Nodes is a special case and deserves more explanation than the other topologies. The main issue with the 14 Nodes topology is the simulations done with a λ equal to 15. These simulations due to being with a higher value of traffic require more bandwidth 70 to work with no dropped packets, yet the delay is more or less the same as the other simulations with different λ. This causes the model to believe that the delay that should be predicted is way higher than the real one, if we look again at the correlation between variables in Figure 5, we see how the bandwidth takes an important role, yet what it is not shown in this table is that in the case where the dataset has no dropped packets, the

bandwidth becomes the most important feature in order to predict the delay. In the case of a λ equal to 15, we have no dropped packets at the cost of increasing the bandwidth, thus the model thinks that the predicted delay should be high.

So in order to not have the results for this topology be relevant I will omit the data from $\lambda = 15$. The table of results with 14 nodes can be found in the appendix on Table 11.

	R_2	ρ	**MAE**	**MRE**	**MSE**
Evaluation	0.9504	0.9873	0.19	−0.09	0.111

Table 9: Table of results for 14 nodes without the data with $\lambda = 15$.

Surprisingly we achieve a lower values of error than the 24 nodes topology, I believe that this is due to the 24 nodes topology having a higher error from high values of delay, which the 14 node topology does not have, if we look at Figure 31 which shows the distribution of delay, we can see how the maximum delay is around 1.6 seconds for that particular dataset, which is a much lower value than the one for the 24 node dataset, as such I believe that this is the cause of the 14 node topology achieving a lower error rate.

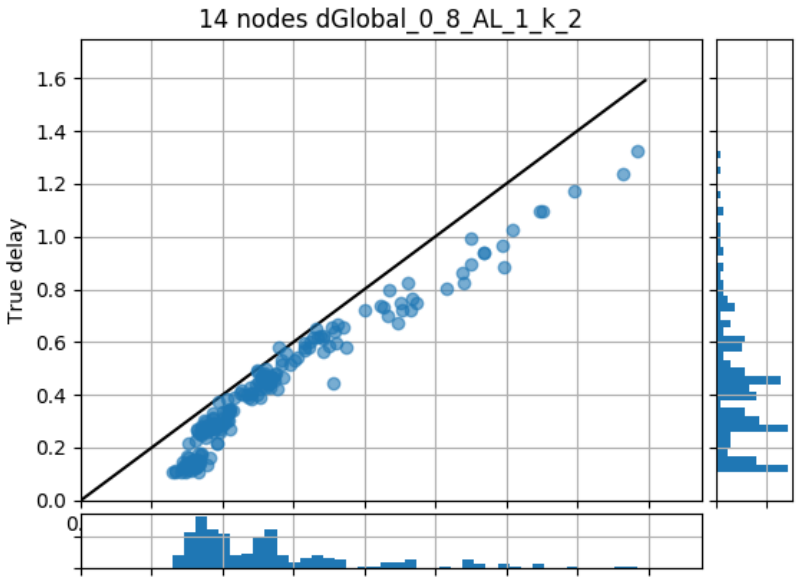


Figure 45: True delay vs Predicted for 14 nodes.

In the case of the 14 nodes, once again RouteNet tends to predict higher values of delay than the real values. But are still reasonable errors. Even more so, if we look at the metrics it is quite clear that the values are good enough to consider the evaluations to be correct.

Due to the small experiment size, with more training routing samples the issues of over optimistic prediction would be solved. Still the only issue in the case of this Figure 45 are the samples lower than 0.2 seconds of delay, which are a bit off of the regression line, the other values, especially the ones between 0.2 and 0.8 are all close to the regression line.

If we look at the previous figures, RouteNet in general seems to have issues trying to predict delays that have a lower values than 0.2 seconds, in both 24 and 50 nodes the predictions do not follow the regression line in the cases where the true delay is lower than 0.2.

5.2.5 Cumulative density function

As a final figure to show the performance, below we can see a cumulative density function for the mean relative error, which shows that the three datasets used for evaluation are indeed quite similar in terms of the mean relative error.

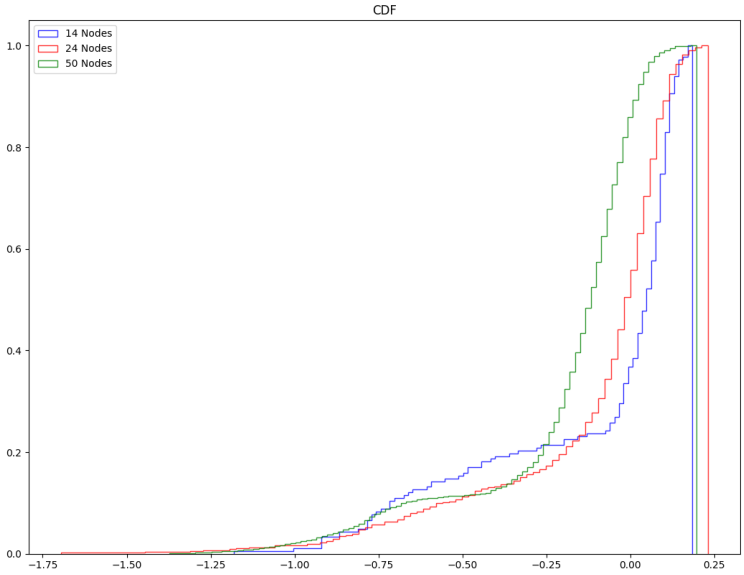


Figure 46: Cumulative density function for mean relative error.

This figure shows how the mean relative error is accumulated, we can see how the majority of the errors are close to 0, meaning that we are achieving a good, low, error for all models. The evaluation for 14 nodes seems to increase faster than the other topologies, meaning that is most likely the worst performing topology of all of them, yet less than 20% of the values have a significant error, which is still an error percentage that it is acceptable. Still the results are good enough that we can say that our model generalizes correctly.

More importantly, it is clear how we have a large amount of negative errors, even thought they are not as numerous, there are a wider range of them, this means that the model, as stated in the previous sections, tends to predict higher values of delay than the real delays.

The table below shows all the metrics for each dataset in a compact way. I have also included the scatter plots for all datasets for easy comparison.

	R_2	ρ	**MAE**	**MRE**	**MSE**
50 Nodes Alternative	0.9944	0.9985	0.02685	0.0786	0.00289
50 Nodes	0.86	0.96	0.047	-0.0377	0.00414
24 Nodes	0.946	0.9861	0.21	−0.115	0.34
14 Nodes	0.9504	0.9873	0.19	−0.09	0.111

Table 10: Result table for all datasets.

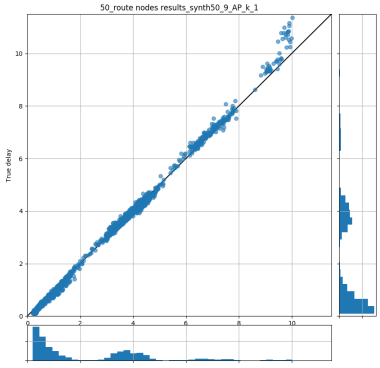


Figure 47: True delay vs Predicted for 50 nodes Alternative.

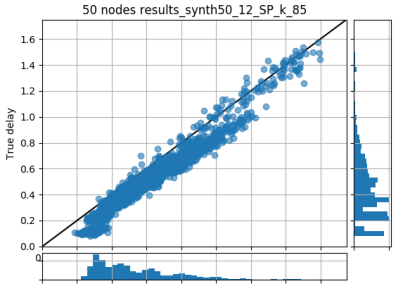


Figure 48: True delay vs Predicted for 50 nodes.

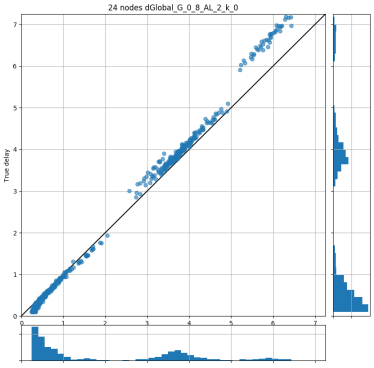


Figure 49: True delay vs Predicted for 24 nodes.

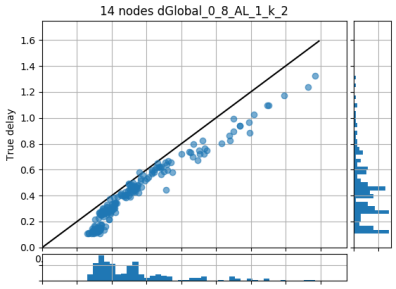


Figure 50: True delay vs Predicted for 14 nodes.

Just as a final comparison, I performed an standard linear model (lm) to have a baseline benchmark for the R^2 metric, this is because on some of the datasets RouteNet achieves great performance, but if there is no other model to compare it to there is no clear way to asses if our model is actually performed good or is it simply that the dataset can be solved using any other standard model. The dataset used for the creation of the *lm* model is the *50 Nodes Alternative*, the model is performed by trying to predict the delay of one routing between two nodes using the delay as the target variable and the rest of attributes for the given source pair as explanatory variables.

The result for model R^2 metric is: **0.6774**, if we compare it to the R^2 value in RouteNet for the same dataset (**0.9944**) it is clear how RouteNet performs way better than a simple baseline model.

6 Conclusions

If we look at the initial objectives of the thesis laid down in the objectives Section 1.4:

1. Create an algorithm that is able to produce a representative sample of routings from a given topology.
2. Prove that in order to obtain a good model it is not needed to use different topologies sizes as train data, but only a representative sample of routings from one topology.
3. Providing a benchmark for Graph Neural Networks for Computer Network Modeling.

In order to check if we have accomplished the first two objectives, we can take a look at the results from the evaluation, especially the values for R^2, which indicates how good is the model at predicting the delay for the network. On all the topologies RouteNet achieves a high value for the R^2 metric, which indicate that RouteNet is indeed able to model the topologies by being trained with routing data from only one topology. Even more so we can make a comparison between the results of our previous paper presented at *SIGCOMM* and the results from this thesis. The best way to compare them would be to use the results for 24 nodes dataset using the two models, one from SIGCOMM and the other from this thesis.

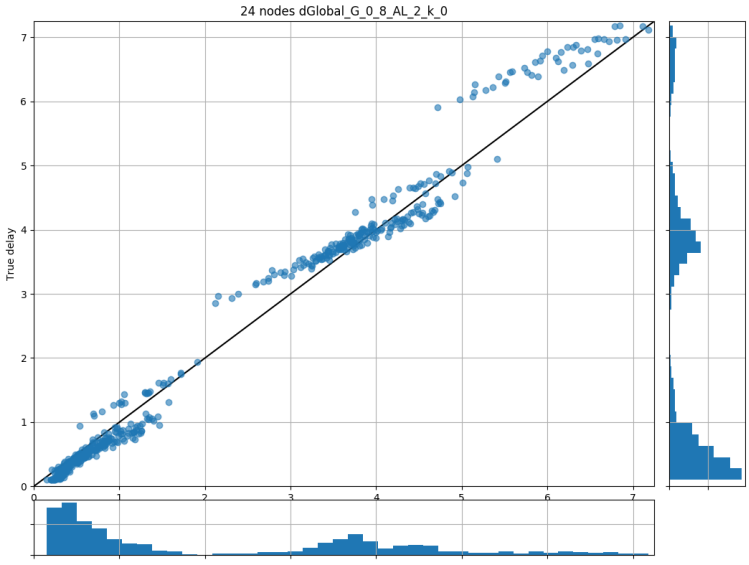


Figure 51: Results from the old model (SIGCOMM) trained with two different topologies and evaluated with the 24 Nodes topology.

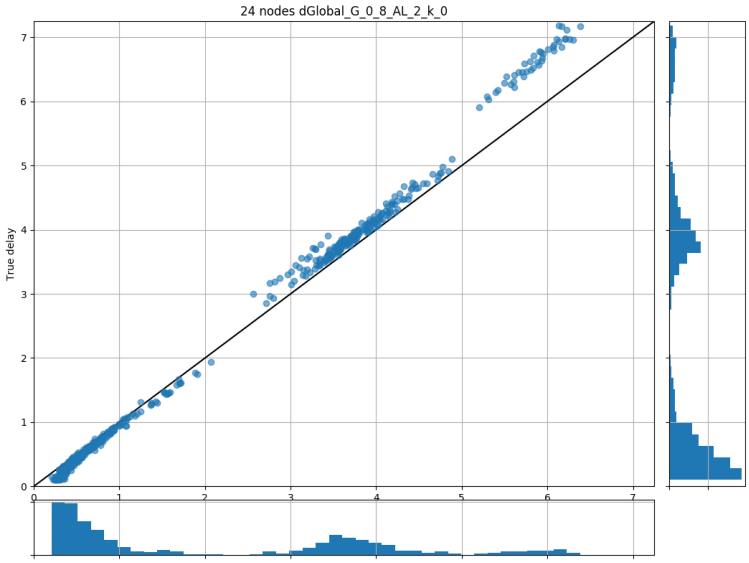


Figure 52: Results for the 24 node topology with the model trained with the alternative routing dataset.

We can see how Figure 52 has the data points closer to regression line, this is an indication that the model is better, or at least equal, to the SIGCOMM model, which once again, is trained with 14 nodes, and 50 nodes topology. Even if we achieve equal results to the SIGCOMM model we can still call the thesis a success, since it means that we are not only able to predict the 24 nodes dataset, but we are able to do it with less training data than the SIGCOMM model. If we take a look at the other results from the evaluation, for 14, and 50 nodes, it is also clear how we are able to achieve good performance results in all topologies.

Even more so, we can compare the results for the 14 Nodes topology, the following scatters plots are one for the SIGCOMM model, using a testing example, and the same example for the new model.

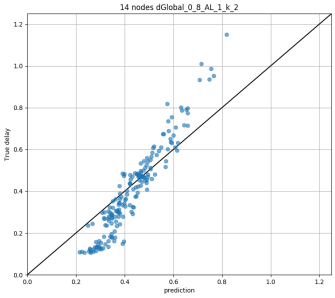


Figure 53: Results for the old model with the 14 Nodes topology

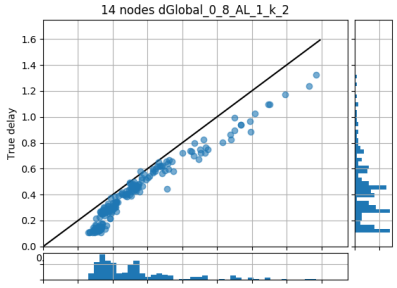


Figure 54: Results for the new model with 14 Nodes Topology.

It is also clear how, judging by the results, I have been able to generate a representative sample of routings using the routing creation algorithm that I implemented.

These results prove that we are indeed able to model any network independently of the node size by training the RouteNet model with only a routing sample from one topology, yet there are some issues that need to be looked at.

- Not predicting properly really high values of delay.
- Unable to produce good predictions in the case of 14 nodes with $\lambda = 15$.
- Incorrect predictions for low values of delay (lower than 0.2 seconds).

The issue with values with high delay is the least worrying of all due to these kinds of values not being realistic, I would consider them not relevant for evaluation proposes. What should be more considered and fixed is the issue with 14 nodes and a high traffic intensity value ($\lambda = 15$), but the issue should be fixed by generating more data to train the model with.

The other issue that needs to be fixed is the one with RouteNet not predicting correctly the delays lower than 0.2 seconds. This is an issue that should be solved by providing more training data, by taking a look at the summary table for the training data set on Figure 29 we can see how the minimum delay is around 0.5 seconds, meaning that the model has not been trained to predict such low values of delay. Adding samples with lower values of average delay would prove very beneficial to RouteNet, and would most likely solve the issue of not being able to predict these kinds of small delays. Both the (1) and (3) problems ca be solved by generating more routing combination, which would mean adding more samples to the dataset.

Reducing the need to train RouteNet with multiple topologies solves the issue of needing to simulate a quite a high number of topologies and training them. This thesis proves that by only using one topology, but extensively exploring the routing possibilities in a single topology produces enough data to train RouteNet which is then able to model other routings from different topologies.

If we take into consideration the size of the experiment done (low amount of samples for training due to time constrains), we can say that overall the results were successful. In conclusion **we are able model any kind of network by only training RouteNet with one topology thanks to being able to generate a representative sample of routings.**

As said in the objectives section in the beginning of this document this means that we do not need to simulate multiple topologies with different sizes, as was stated in the SIGCOMM paper, thus reducing the amount of time to train a new RouteNet model.

For the last objective, the creation of a benchmark, the model and the datasets used and created for this thesis can be found , which are publicly available for anyone to try and improve. [35] [36]

6.1 Future Work

In order to produce better results, more data would be needed, mainly to work with the edge cases as well as reducing the error, especially for the 24 and 14 nodes topology, whose errors are a bit too high even though the results for R^2 and ρ are good. This indicates a good performing model, but due to a lack of data on certain delay values, as explained before, the error value increases. Also it is likely that changes in the simulation would be needed in order to produce more realistic data, for example, all the traffic in the system is more or less the same. The number of packets is also usually the same also.

Simulations for topologies with a higher amount of nodes would also be needed, the higher the node degree the more paths are possible to generate, thus producing better, and more numerous, training samples. It would also be needed since, as said in the introduction, I believe that RouteNet is able to model any network provided the network size is similar or lower than the training network topology size.

In order to explore the topologies even more to create more routing combinations, which are needed in order to obtain even better results by creating a model which is able to generalize routings even more, the problem of creating valid routings need to be solved. As of right now all the routings are combinations of shortest path routings, yet it is preferable to create routing schemes that are a combination of shortest path, and non shortest path. To do so the

simulation program needs to be changed to support source-destination routing, which allows to send packets in which the source of the packets is taken into consideration in order to deliver the packet. This would solve the issues of having loops in the routing creation.

Technologies to handle large amounts of data for training and evaluation would also be useful.

Overall we can say that the thesis provides the ground work for future research on the topic of graph neural networks for computer networking.

7 References

References

[1] Peter W. Battaglia et al. "Relational inductive biases, deep learning, and graph networks". In: (2018), pp. 1–40. ISSN: 0031-1820. DOI: 10.1017/S0031182005008516. arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

[2] Giorgio Stampa et al. *A Deep-Reinforcement Learning Approach for Software-Defined Networking Routing Optimization*. 2017. arXiv: 1709.07080. URL: http://arxiv.org/abs/1709.07080.

[3] Albert Mestres et al. "Knowledge-Defined Networking". In: *CoRR* abs/1606.0 (2016). arXiv: 1606.06222. URL: http://arxiv.org/abs/1606.06222.

[4] N Wang et al. "An overview of routing optimization for internet traffic engineering". In: *IEEE Communications Surveys Tutorials* 10.1 (2008), pp. 36–56. ISSN: 1553-877X. DOI: 10.1109/COMST.2008.4483669.

[5] B Fortz and M Thorup. "Internet traffic engineering by optimizing OSPF weights". In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 2. 2000, 519–528 vol.2. DOI: 10.1109/INFCOM.2000.832225.

[6] 2019. URL: http://conferences.sigcomm.org/sigcomm/2019/.

[7] 2019. URL: https://dawn.cs.stanford.edu/benchmark/ImageNet/train.html.

[8] Krzysztof Rusek et al. *Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN*. 2019. arXiv: 1901.08113 [cs.NI].

[9] A. Graves et al. "A Novel Connectionist System for Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5 (2009), pp. 855–868. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137.

[10] Xiangang Li and Xihong Wu. "Constructing Long Short-Term Memory based Deep Recurrent Neural Networks for Large Vocabulary Speech Recognition". In: *CoRR* abs/1410.4281 (2014). arXiv: 1410.4281. URL: http://arxiv.org/abs/1410.4281.

[11] Justin Gilmer et al. "Neural Message Passing for Quantum Chemistry". In: (2017). ISSN: 00295981. DOI: 10.1002/nme.2457. arXiv: 1704.01212. URL: http://arxiv.org/abs/1704.01212.

[12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012. ISBN: 026201825X, 9780262018258.

[13] S J Russell et al. "Artificial Intelligence: A Modern Approach". In: vol. 74. Jan. 1995.

[14] Tom M Mitchell. *The need for biases in learning generalizations*. Department of Computer Science, Laboratory for Computer Science Research . . ., 1980.

[15] OpenSim Ltd. *OMNeT++ Discrete Event Simulator*. `https://omnetpp.org/`. 2018.

[16] Fernando Barreto. "Fast Emergency Paths Schema to Overcome Transient Link Failures in OSPF Routing". In: *International journal of Computer Networks Communications* 4.2 (2012), 17–34. ISSN: 0975-2293. DOI: `10.5121/ijcnc.2012.4202`. URL: `http://dx.doi.org/10.5121/ijcnc.2012.4202`.

[17] Xiaojun Hei et al. "Wavelength converter placement in least-load-routing-based optical networks usinggenetic algorithms". In: *J. Opt. Netw.* 3.5 (2004), pp. 363–378. DOI: `10.1364/JON.3.000363`. URL: `http://jon.osa.org/abstract.cfm?URI=jon-3-5-363`.

[18] Renaud Hartert et al. "A declarative and expressive approach to control forwarding paths in carrier-grade networks". In: *ACM SIGCOMM computer communication review* 45.4 (2015), pp. 15–28.

[19] Thomas H. Cormen et al. *Introduction to Algorithms*. 2nd. Cambridge, MA, USA: MIT Press, 2001. ISBN: 0-262-03293-7, 9780262032933.

[20] 2019. URL: `https://docs.python.org/3/reference/`.

[21] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by Gaël Varoquaux, Travis Vaught, and Jarrod Millman. Pasadena, CA USA, 2008, pp. 11 –15.

[22] 2019. URL: `https://www.tensorflow.org/`.

[23] 2019. URL: `https://www.tensorflow.org/guide/summaries_and_tensorboard`.

[24] François Chollet. *FAQ - Keras Documentation*. 2019. URL: `https://keras.io`.

[25] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: `1406.1078`. URL: `http://arxiv.org/abs/1406.1078`.

[26] Junyoung Chung et al. "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling". In: *CoRR* abs/1412.3555 (2014). arXiv: `1412.3555`. URL: `http://arxiv.org/abs/1412.3555`.

[27] 2019. URL: `http://www.wildml.com/2015/10/recurrent-neural-network-tutorial-part-4-implementing-a-grulstm-rnn-with-python-and-theano/`.

[28] 2019. URL: https://www.d2l.ai/chapter_recurrent-neural-networks/gru.html.

[29] Günter Klambauer et al. "Self-Normalizing Neural Networks". In: *CoRR* abs/1706.02515 (2017). arXiv: 1706.02515. URL: http://arxiv.org/abs/1706.02515.

[30] F. Santosa and W. Symes. "Linear Inversion of Band-Limited Reflection Seismograms". In: *SIAM Journal on Scientific and Statistical Computing* 7.4 (1986), pp. 1307–1330. DOI: 10.1137/0907087. eprint: https://doi.org/10.1137/0907087. URL: https://doi.org/10.1137/0907087.

[31] Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning". In: *international conference on machine learning*. 2016, pp. 1050–1059.

[32] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. arXiv: 1412.6980 [cs.LG].

[33] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. 2019. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[34] François Chollet. *FAQ - Keras Documentation*. 2019. URL: https://keras.io/getting-started/faq/#why-is-the-training-loss-much-higher-than-the-testing-loss.

[35] 2019. URL: http://knowledgedefinednetworking.org/.

[36] Sergi Carol Bosch. *Model for benchmark*. 2019. URL: https://github.com/SergiCarol/RouteNet-Benchmark/tree/master/Code.

8 Appendix

8.1 True Delay vs Predicted 50 Nodes alternative

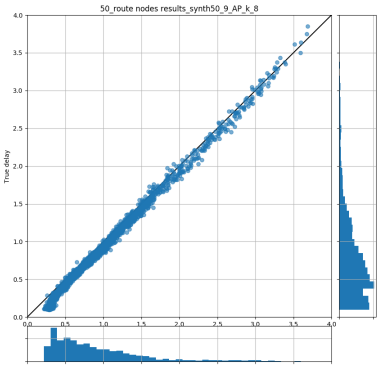


Figure 55: λ 9 routing 8

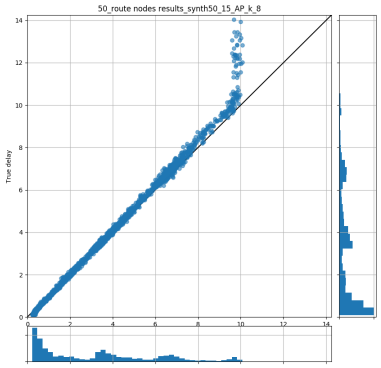


Figure 56: λ 15 routing 8

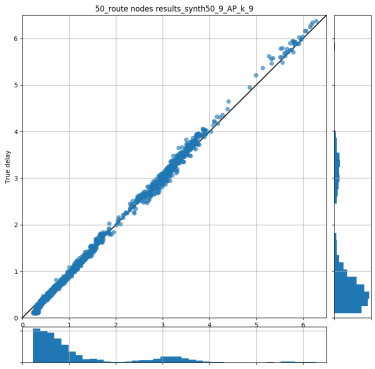


Figure 57: λ 9 routing 9

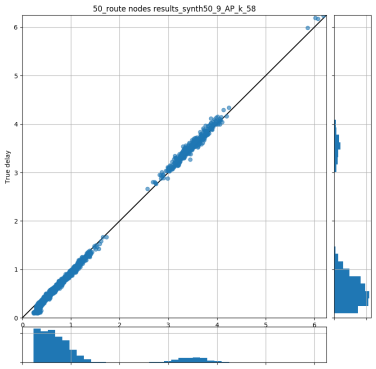


Figure 58: λ 9 routing 58

8.2 True Delay vs Predicted 50 Nodes

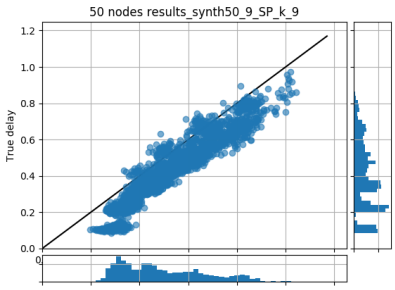


Figure 59: λ 9 routing 9

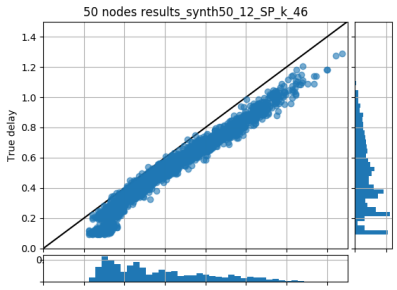


Figure 60: λ 12 routing 46

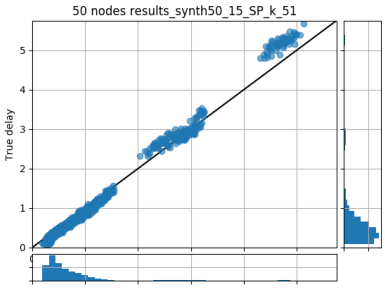


Figure 61: λ 15 routing 51

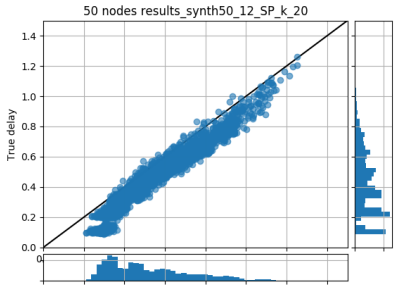


Figure 62: λ 12 routing 20

8.3 True Delay vs Predicted 24 Nodes

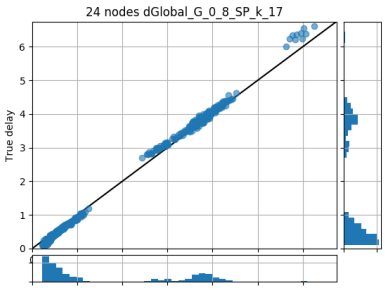


Figure 63: λ 8 routing 17

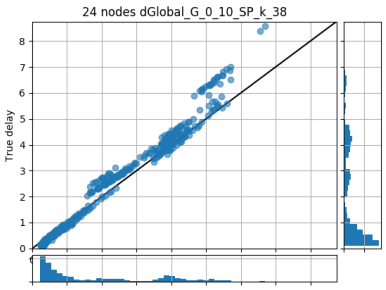


Figure 64: λ 10 routing 38

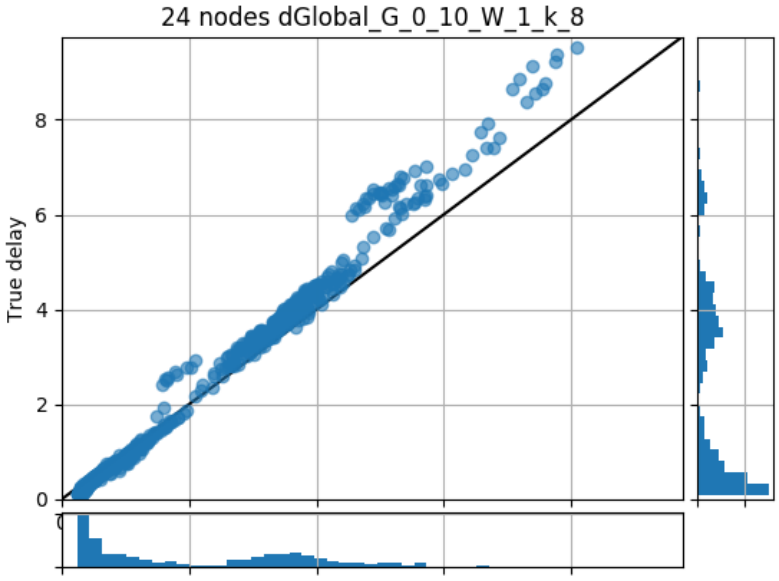


Figure 65: λ 10 routing 1 k 8

8.4 True Delay vs Predicted 14 Nodes

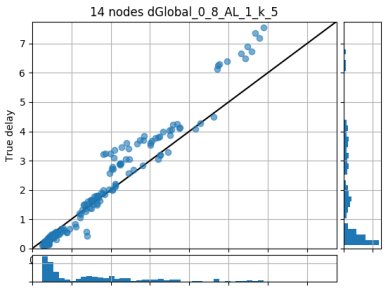


Figure 66: λ 8 routing AL 1 k 5

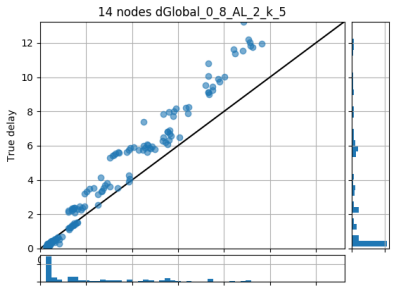


Figure 67: λ 8 routing 2 k 5

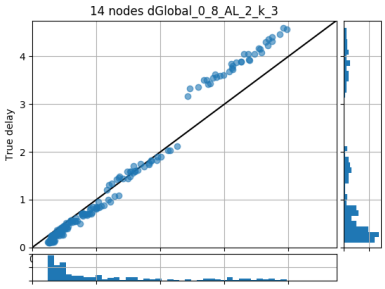


Figure 68: λ 8 routing AL 2 k 3

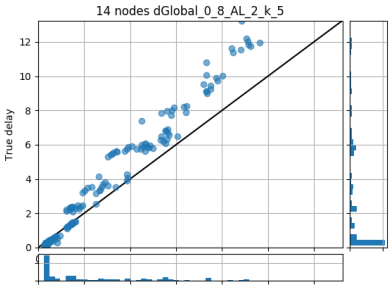


Figure 69: λ 8 routing 2 k 5

8.5 Table of results for 14 Nodes with λ 15

	R^2	ρ	**MSE**	**MRE**	**MAE**
14 Nodes Full	0.23	0.49	1.80	0.161	0.71

Table 11: Results for 14 nodes with λ 15 included in the dataset

8.6 Bandwidth for 14 nodes with λ 15

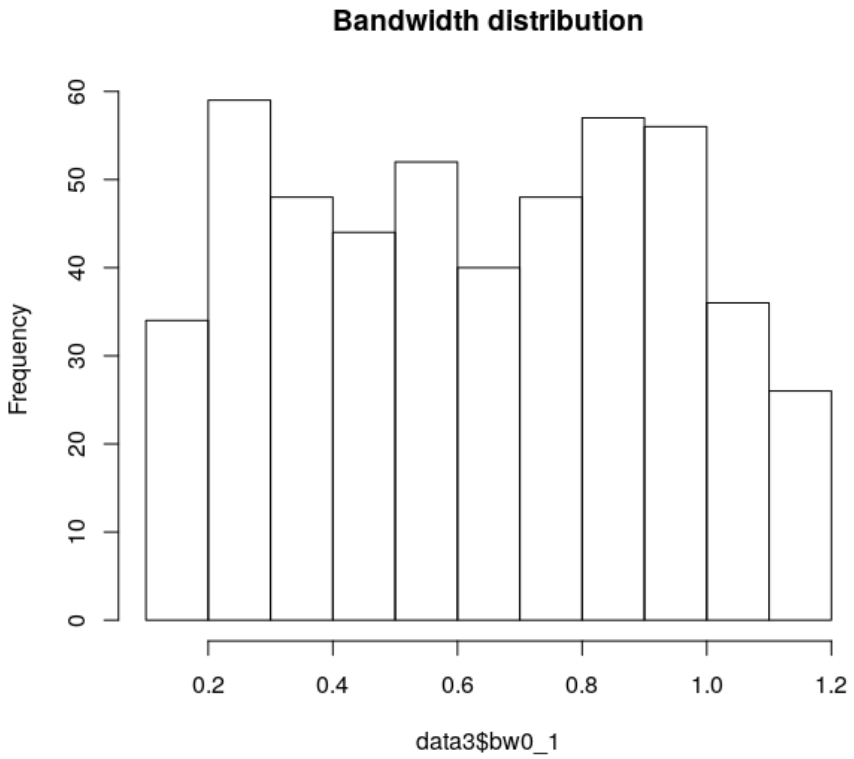


Figure 70: 14 Nodes lambda 15 bandwidth

8.7 PCA results

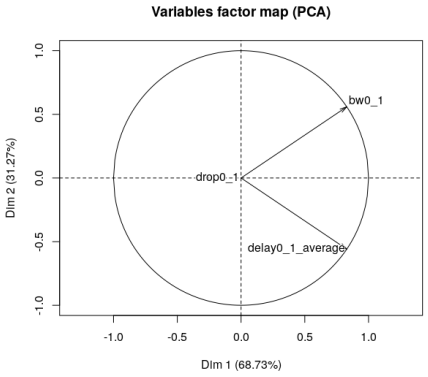


Figure 71: 14 Nodes

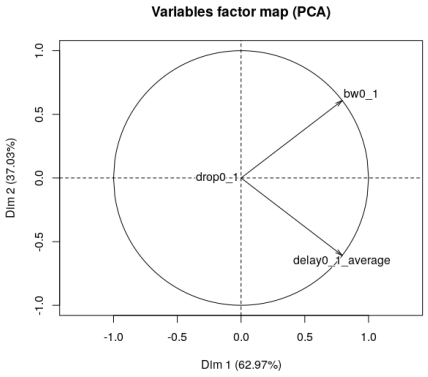


Figure 72: 24 Nodes

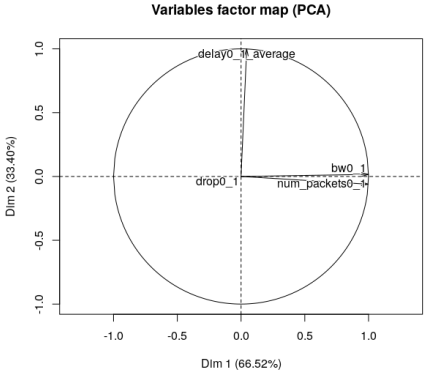


Figure 73: 50 Nodes

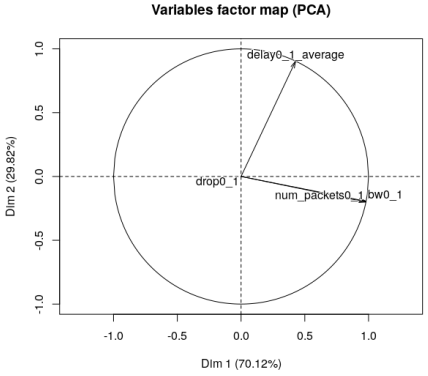


Figure 74: 50 Nodes Alternative

8.8 Model Performance

In this section I will evaluate the model performance by showing a variety of graphs that asses how well the model is created.
The figures below show the evolution of the distribution of values that come as an input, Figure 75, and the output, Figure 76.

From the graphs it easily to see how the distribution from the input and the output is similar over the training steps.

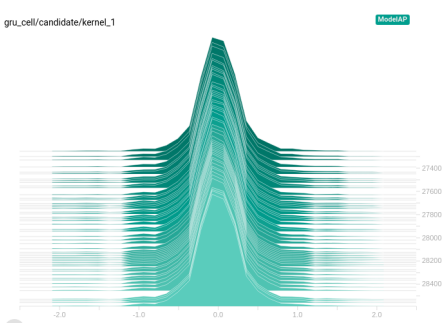


Figure 75: GRU Kernel candidate distribution.

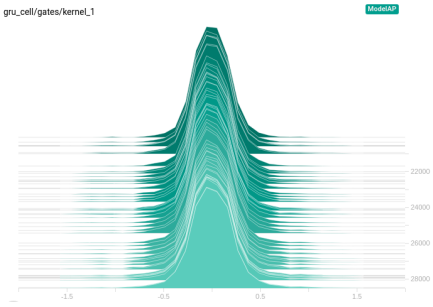


Figure 76: GRU gate value distribution, notice the change on the scale of the x axis.